University of Castilla-La Mancha

A publication of the

Computing Systems Department

Hardware implementation study of several new egress link scheduling algorithms

by

Raúl Martínez, Francisco J. Alfaro, José L. Sánchez, José Manuel Claver

Technical Report **#DIAB-08-07-1** July 2008

COMPUTING SYSTEMS DEPARTMENT
COMPUTER SCIENCE SCHOOL
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

Hardware implementation study of several new egress link scheduling algorithms

R. Martínez, F.J. Alfaro, J.L. Sánchez

Dpto. de Sistemas Informáicos
Univ. Castilla-La Mancha
Albacete, SPAIN 02071
rmartinez,falfaro,jsanchez@dsi.uclm.es

J.M. Claver

Dpto de Informáica
Univ. Valencia
Valencia, SPAIN 46100
jclaver@uv.es

***Abstract*— The provision of Quality of Service (QoS) in computing and communication environments has been the focus of much discussion and research in academia during the last decades. This interest in academia has been renewed by the growing interest on this topic in industry during the last years. A key component for networks with QoS support is the egress link scheduling algorithm. Apart from providing a good performance in terms of, for example, good end-to-end delay (also called latency) and fair bandwidth allocation, an ideal scheduling algorithm implemented in a high-performance network with QoS support should satisfy other important property which is to have a low computational and implementation complexity.**

In this paper, we propose specific implementations (taking into account the characteristics of current high performance networks) of several fair-queuing scheduling algorithms and compare their complexity in terms of silicon area and computation delay. In order to carry out this comparison, we have performed our own hardware implementation for the different schedulers. We have modeled the schedulers using the Handel-C language and employed the DK design suite tool from Celoxica in order to obtain hardware estimates on silicon area and arbitration time.

***Index Terms*— Scheduling algorithms, interconnection networks, Quality of Service, Advanced Switching, InfiniBand, hardware implementation, complexity stimation.**

I. Introduction

The evolution of the interconnection network technology has been constant along the previous decades. The speed and capacity of various components in a communication system, such as links, switches, memory, and processors, have increased dramatically. Moreover, network topologies have become more flexible, and the efficiency of switching, routing and flow control techniques have been improved.

The advent of high-speed networking has introduced opportunities for new applications. Current packet networks are required to carry not only traffic of applications, such as e-mail or file transfer, which does not require pre-specified service guarantees, but also traffic of other applications that requires different performance guarantees, like real-time video or telecommunications [20]. The best-effort service model, though suitable for the first type of applications, is not so for applications of the other type [23]. Even in the same application, different kinds of traffic (e.g. I/O requests, coherence control messages, synchronization and communication messages, etc.) can be considered, and it would be very interesting that they were treated according to their priority [10].

The provision of QoS in computing and communication environments has been the focus of much discussion and research in academia during the last decades. This interest in academia has been renewed by the growing interest on this topic in industry during the last years. A sign of this growing interest in industry is the inclusion of mechanisms intended for providing QoS in some of the last network standards like Gigabit Ethernet [26], InfiniBand [15], or Advanced Switching (AS) [1]. An interesting survey with the QoS capabilities of these network technologies can be found in [24].

A common characteristic of the specifications of

This work has been jointly supported by the Spanish MEC and European Comission FEDER funds under grants Consolider Ingenio-2010 CSD2006-00046 and TIN2006-15516-C04-0X and by Junta de Comunidades de Castilla-La Mancha under grant PCC08-0078-9856.

these network technologies intended to provide QoS are the use of a reduced set of Virtual Channels (VCs) and an egress link scheduler to arbitrate among the traffic transmitted in each VC. These mechanisms permit us to aggregate traffic with similar characteristics in the same VC and to provide each VC with a different treatment according to its requirements, at the stile of the differentiated services (DiffServ) QoS model [6], [5].

A key component for networks with QoS support is the output (or egress link) scheduling algorithm (also called service discipline) [11], [14], [37]. In a packet-switching network, packets from different flows will interact with each other at each switch. Without proper control, these interactions may adversely affect the network performance experienced by clients. The scheduling algorithm, which selects the next packet to be transmitted and decides when it should be transmitted, determines how packets from different flows interact with each other. Therefore, the scheduling algorithm plays an important role to perform the traffic differentiation that is necessary to provide QoS.

Apart from providing a good performance in terms of, for example, good end-to-end delay (also called latency) and fair bandwidth allocation, an ideal scheduling algorithm implemented in a high-performance network with QoS support should satisfy other important property which is to have a low computational and implementation complexity [28]. This is because in order to achieve a good performance, the time required to select the next packet to be transmitted must be smaller than the average packet transmission time. This means that the scheduler computation time must be very small, if we consider the high speed of high-performance networks. Moreover, a low complexity is required in order to be able to implement the scheduler in a small silicon area (note that high-performance switches are usually implemented in a single chip).

The design of a traffic scheduling algorithm involves an inevitable trade-off among the above properties. Many scheduling algorithms have been proposed for that. Among them, the “sorted-priority” family of algorithms are known to offer very good delay [31]. However, their computational complexity is very high, making their implementation in high-speed networks rather difficult. In order to avoid the complexity of the sorted-priority approach, the Deficit Round Robin (DRR) algorithm [27] has been proposed. The main problem of this algorithm is that depending on the situation the latency can be very bad.

On the other hand, table-based schedulers are intended to provide a good latency performance with a low computational complexity. This approach is followed in [9] and in two of the last high-performance network interconnection proposals: Advanced Switching (AS) [1] and InfiniBand (IBA) [15]. However, as we will see, these schedulers do not work properly with variable packet sizes, as is usually the case in current network technologies. This is the reason because in [17] we proposed a new table-based scheduler that works properly with variable packet sizes. Moreover, we proposed a methodology to configure this scheduler in such a way that it permits us to decouple partially the bounding between the bandwidth and latency assignments. We called this new scheduler Deficit Table scheduler, or just DTable scheduler.

We can measure the complexity of a scheduler based on two parameters: Silicon area required to implement the scheduling mechanism and time required to determine the next packet to be transmitted. A short scheduling time is an efficiency requirement. The next packet to be transmitted should be chosen during the transmission time of the last packet which was selected by the scheduler. This is necessary in order to be able to send packets one after another without letting gaps between them. This requirement takes more importance in high-performance networks due to their high speed. Moreover, switches of high-performance interconnection technologies are usually implemented in a single chip. Therefore, the silicon area required to implement the various switch elements is a key design feature. Note, that a scheduling algorithm must be implemented in each egress link and thus, the silicon area required to implement the scheduling algorithm should be as small as possible.

In this paper, we propose specific implementations (taking into account the characteristics of current high performance networks) of several fair-queuing scheduling algorithms and compare their complexity in terms of silicon area and computation delay. The scheduling algorithms that we have chosen for this comparison are the Self-Clocked Fair Queuing (SCFQ) [13], the DRR, and the DTable. We have chosen the SCFQ algorithm as an example of “sorted-priority” algorithm, the DRR algorithm

because of its very small computational complexity, and the DTable as an intermediate proposal. In the case of the SCFQ and DRR schedulers, we use the credit aware versions of both algorithms that we proposed in [18] for being used in networks with a link-level flow control network, which is the case in most current high-performance network technologies.

In [33] and [25] interesting implementations for the SCFQ scheduler are proposed. However, these implementations were designed for a high number of possible flows. Note that in our case there is going to be just a limited number of VCs. This allows to consider more efficient implementations. Moreover, the case of the SCFQ implementation [25] was intended for fixed packet sizes, specifically, for an ATM environment.

Therefore, we have performed our own hardware implementation for the different schedulers. We have modeled the schedulers using the Handel-C language [7] and employed the DK design suite tool from Celoxica in order to obtain hardware estimates on silicon area and arbitration time.

The structure of the paper is as follows: Section II presents a summary of the general aspects about scheduling algorithms, focusing on the fair queuing schedulers family. Sections III, IV, and V present the DRR-CA, SCFQ-CA, and DTable scheduling algorithms, respectively, and describe their hardware implementation. In Section VI a comparison study on the implementation and computational complexity of the different schedulers is provided. Finally, some conclusions are given.

II. Scheduling algorithms

Service discipline, also called packet scheduling, is an important mechanism to provide QoS guarantees in computer networks, such as end-to-end delay bounds and fair bandwidth allocation [11], [14], [37]. During the last decades a vast amount of scheduling disciplines have been proposed in the literature for different purposes. This section outlines some desirable properties of scheduling disciplines and presents possible ways to classify scheduling disciplines.

In order to be able to design new scheduling disciplines and to compare the existing ones with each other, it is important to define the desirable properties of a scheduling discipline. It is obvious that many of these properties are tightly related to the QoS guarantees made for the end user. However, there are also some general desirable properties:

1) **Good End-to-End Delay**. As stated before, the end-to-end delay (also called latency) is defined as the sum of the transmission delay, the propagation delay, and the queuing delay experienced at each network node. The last component is by far the most significant. In some applications if a packet experiences a latency higher than a certain value, the value of the packet information may be greatly diminished or even worthless. Moreover, a larger delay bound implies increased burstiness of the session at the output of the scheduler, thus increasing the buffering needed at the switches to avoid packet losses [31]. Thus, a good scheduling algorithm should guarantee acceptable queuing delay.
2) **Flexibility**. The scheduling discipline should be able to accommodate applications with varying traffic characteristics and performance requirements rather than just optimize the performance from a certain application's point of view [37]. In future networks several applications with diverse requirements will have to be supported making necessary for the scheduling discipline to be flexible.
3) **Protection**. Real network environment is not static. As a consequence, the scheduling discipline should be able to protect the well behaving users from different sources of variability, such as best-effort traffic, bad behaving users and network load fluctuations [37]. Bad behaving users refer, for example, to users who send more packets than their traffic profile allows. Network load fluctuations, on the other hand, are caused by traffic bursts at a router. These bursts may accumulate even if the users meet their traffic constraints at the entrance of the network. Ideally, the scheduling discipline should be able to satisfy the performance requirements of well behaving users even in the presence of these factors.
4) **Simplicity**. Performance characteristics are not the only parameters that must be taken into account when deciding which is the best scheduler in networks with QoS support. Other important property, specially in high-

performance networks, is simplicity [28]. This is because in order to achieve a good performance, the processing overheads must be some orders of magnitude smaller than the average packet transmission time. This means that the time needed to decide the next packet to be transmitted must be very small, if we consider the high speed of high-performance networks. Moreover, a low complexity is required in order to be able to implement the scheduler in a small silicon area (note that high-performance switches are usually implemented in a single chip).

Scheduling disciplines can be categorized in many ways. Traditionally they have been divided into work-conserving and non-work-conserving disciplines [37]. Another possible classification is based on their internal structure, according to two main architectures: Sorted-priority and frame-based [29].

A well-known and very important kind of scheduling algorithms are the fair queuing algorithms. This kind of algorithms allocate bandwidth to the different flows in proportion to a specified set of weights. The perfect fair queuing scheduling algorithm is the General Processor Sharing (GPS) scheduler [11], [22].

GPS is considered to be an attractive scheduling discipline since it has many desirable properties. First, it provides fairness for the flows by servicing each flow with a rate equal to or greater than the flows's guaranteed rate. Second, if the incoming traffic is leaky-bucket constrained [32], it has been proved that strict bounds for worst-case network queuing delay exist [22]. Third, the classes can be treated in different ways by varying the weights. For instance, if there are two classes with weights $\phi_1 = 1$ and $\phi_2 = 0$, GPS reduces to strict priority scheduling. On the other hand, if all classes are assigned equal weights, GPS behaves as a uniform processor sharing system.

However, despite these advantages, GPS is not a realistic service discipline since in a packet network service it is performed packet-by-packet, rather than "bit-by-bit" and thus, it cannot be implemented in practice. Different packet-by-packet approximations of GPS have been proposed, which try to emulate the GPS system as accurately and simply as possible while still treating packets as entities. Examples of fair queueing algorithms are Weighted Fair Queuing (WFQ) [11], packet-by-packet GPS [22], Self-Clock Fair Queueing SCFQ [13], Worst Case Weighted Fair Queuing (WF2Q) [3], frame-based fair queuing [30], Hierarchical Packet Fair Queuing [4], Weighted Round Robin (WRR), Deficit Round Robin (DRR) [27], and List-based WRR [9]. These scheduling algorithms can be divided in two big groups: Sorted-priority algorithms and frame-based algorithms.

A. Sorted-priority fair queuing algorithms

A real-world packet-by-packet GPS service discipline typically consists of the following two functions:

1) **Tracking GPS time:** This function tracks the progress of GPS virtual time (described later) with respect to the real time. Its main objective is to estimate the GPS virtual start and finish times of a packet, which are the times that a packet should have started and finished to be served, respectively, if served by a GPS scheduler.
2) **Scheduling according to GPS clock:** This function schedules the packets based on the estimation of their GPS virtual finish/start times. For example, WFQ selects the packet with the lowest GPS virtual finish time among the packets currently in queue to be served.

The algorithms that follow this approach are included in the "Sorted-priority" family of algorithms. This kind of scheduling algorithms assign each packet a tag and scheduling is made based on the ordering of these tags. "Sorted-priority" algorithms are known to offer good delay bounds [31]. However, this family of algorithms suffers from two major problems. The first problem is that these algorithms require processing at line speeds for tag calculation and tag sorting. In other words, each time a packet arrives at a node, its time tag is calculated and the packet is inserted at the appropriate position in the ordered list of packets waiting for transmission. This means that these algorithms require at least the complexity of a search algorithm in the list of queued packets: $O(log(N))$, where N is the maximum number of packets at the queue, or if the buffers are not shared, $O(log(J))$, where J is the number of active flows. The complexity of computing the GPS virtual finish time of the packets has long been believed to be $O(J)$ [22], [30], [31],

[8]. In [38] and [36] a deeper discussion on this topic can be found.

The second problem that may happen in the sorted-priority approach is that the virtual clock cannot be reinitialized to zero until the system is completely empty and all the sessions are idle. The reason is that the time tag is an increasing function of the time and depends on a common-reference virtual clock, which in turns reflects the value of the time tag of previously served packets. In other words, it is impossible to reinitialize the virtual clock during the busy period, which, although statistically finite (if the traffic is constrained), can be extremely long, especially given that most communication traffic has been shown to exhibit self-similar patterns which lead to heavily tailed buffer occupancy distributions.

Therefore, for a practical implementation of sorted-priority algorithms, very high-speed hardware needs to be designed to perform the sorting, and floating-point units must be involved in the computation of the time tags.

B. Frame-based fair queuing algorithms

Frame-based fair queuing algorithms try to address the excesive complexity of sorted-priority algorithms. The simplest frame-based scheduling discipline that provides a way to emulate the GPS system is the Weighted Round Robin (WRR). In the WRR approach, a list of flow weights is visited sequentially, each weight indicating the number of packets from the flow that can be transmitted. The WRR algorithm faces a problem if the average packet size of the different flows is different. In that case, the bandwidth that the flows obtain may not be proportional to the assigned weights. Therefore, the WRR algorithm does not work properly with variable packet sizes. However, today network technologies usually use variable packet sizes.

The Deficit Round Robin (DRR) algorithm [27] is a variation of the WRR algorithm that works on a proper way with variable packet sizes. In order to handle properly variable packet sizes, the DRR algorithm associates each queue with a *quantum* and a *deficit counter*. The quantum assigned to a flow is proportional to the bandwidth assigned to that flow. The sum of all the quantums is called the frame length. For each flow, the scheduler transmits as many packets as the quantum allows. When a packet is transmitted, the quantum is reduced by the packet size. The unused quantum is saved in the deficit counter, representing the amount of quantum that the scheduler owes the flow. At the next round, the scheduler will add the previously saved quantum to the current quantum. The main advantage of the DRR scheduler is its computational simplicity. Due to this, recent research in the Differentiated Services area proposes the DRR as a feasible solution for implementing the Expedited Forwarding Per-hop Behavior [12]. However, the main problem of this algorithm is that its delay depends on the frame length. Depending on the situation, the frame can be very long, and thus, the latency could be very bad.

Two recent network technology standards, AS and IBA, incorporate table-based schedulers, which are intended to provide a good latency performance with a small computational complexity. In order to provide a good latency performance, the table-based schedulers instead of serving packets of a flow in a single visit per frame, the service is distributed throughout the entire frame. AS and IBA use Virtual Channels (VCs) to aggregate flows with similar characteristics and the arbitration is made at a VC level. In both cases, the maximum number of unicast VCs that a port can implement is 16. The AS table-based scheduler employs an arbitration table that consists in a register array with fixed-size entries of 8 bits. Each entry contains a field of 5 bits with a VC identifier value and a reserved field of 3 bits. When arbitration is needed, the table is cycled through sequentially and a packet is transmitted from the VC indicated in the current table entry regardless of the packet size. If the current entry points to an empty VC, that entry is skipped. The number of entries may be 32, 64, 128, 256, 512, or 1024.

InfiniBand defines a scheduler that uses two tables, one for scheduling packets from high-priority VCs and another one for low-priority VCs. The maximum amount of data that can be transmitted from high-priority VCs before transmitting a packet from the low-priority VCs can be configured. Each table has up to 64 entries. Each entry contains a VC identifier and a weight, which is the number of units of 64 bytes to be transmitted from that VC. This weight must be in the range of 0 to 255, and is always rounded up as a whole packet.

On the other hand, Chaskar and Madhow [9] propose a category of scheduler called list-based

Weighted Round Robin for being used in networks with fixed packet sizes. Chaskar and Madhow propose three of these list-based WRR schedulers. All of these schedulers can actually be implemented with the AS table-based scheduler. In all the cases the proportion of table entries associated with each flow indicates the bandwidth assigned to each flow. Therefore, the difference among the three schedulers is in the way of distributing the flow identifiers among the table entries. These different forms of interleaving the flow identifiers result in different latency characteristics for the three schedulers. In their tests, the variant that provides the best latency performance tries to emulate the behavior of the WF2Q algorithm.

III. The DRR-CA scheduler

The DRR algorithm [27] is a variation of the WRR algorithm that works on a proper way with variable packet sizes. In order to handle properly variable packet sizes, the DRR algorithm associates each queue with a *quantum* and a *deficit counter*. The quantum assigned to a queue is proportional to the bandwidth assigned to that queue. The deficit counter is set to 0 at the beginning. The scheduler visits sequentially each queue and transmits as many packets as the quantum allows. When a packet is transmitted, the quantum is reduced by the packet size. The unused quantum is saved in the deficit counter, representing the amount of quantum that the scheduler owes the queue. At the next round, the scheduler will add the previously saved quantum to the current quantum. When the queue has no packets to transmit, the quantum is discarded, since the flow has wasted its opportunity to transmit packets. Figure 1 shows the pseudocode for this algorithm.

The problem of the DRR scheduler, which is common in current high-peformance networks, is that it interacts with the link-level flow control mechanism. When we do not allow the selection of a flow, because of lack of flow control credits, if we still continue accumulating quantum for this flow in each round, then the blocked flow is going to take advantage of the time that has been blocked. In order to solve this problem, the DRR-CA algorithm that we have proposed works in the same way as the DRR algorithm, except in the following aspects:

- We are going to consider VCs instead of flows in the DRR-CA algorithm because, as stated before, is the common trend in the last interconnection network proposals.
- A VC queue is considered active only if it has at least one packet to transmit and if there are enough credits to transmit the packet at the head of the VC.
- When a packet is transmitted, the next active VC is selected when any of the following conditions occurs:
  - There are no more packets from the current VC or there are not enough flow control credits for transmitting the packet that is at the head of the VC. In any of these two cases, the current VC becomes inactive, and its deficit counter becomes zero.
  - The remaining quantum is less than the size of the packet at the head of the current VC. In this case, its deficit counter becomes equal to the accumulated weight in that instant.

The resulting algorithm is expressed in the pseudocode shown in Figure 2.

If we compare the complexity of the DRR and DRR-CA algorithms, the main difference is that in the case of the DRR-CA algorithm the number of queues is equal to the number of VCs instead of the number of flows, and thus the complexity is even smaller. The only added complexity is to take into account the flow control status in order to consider active or inactive a VC.

A well-known problem of the WRR and DRR algorithms, that is also shared by the DRR-CA algorithm, is that the latency and fairness depend on the frame length. The frame length in these algorithms is defined as the sum of all the weights in the WRR algorithm or the quantums in the DRR algorithm. The longer the frame is, the higher the latency and the worse the fairness. In order for DRR to exhibit lower latency and better fairness, the frame length should therefore be kept as small as possible. Unfortunately, given a set of flows, it is not possible to select the frame length arbitrarily. According to the implementation proposed in [27], DRR exhibits $O(1)$ complexity provided that each flow is allocated a quantum no smaller than the Maximum Transfer Unit (MTU). This ensures that the algorithm can cycle through the list knowing that it is always possible to transmit at least one packet

```
while (There is at least one packet to be transmitted)
    if ((There are no packets in the queue of selectedFlow) or
        (selectedFlow_sizeFirst > totalQuantum))
        deficitCounter_selectedFlow ← totalQuantum
        selectedFlow ← Next active flow
        totalQuantum ← deficitCounter_selectedFlow + quantum_selectedFlow
    totalQuantum = totalQuantum − selectedFlow_sizeFirst
    Transmit packet from selectedFlow
    if (There are no more packets in the queue of selectedFlow)
        totalQuantum ← 0
```

Fig. 1. Pseudocode of the DRR scheduler.

```
while (There is at least one active VC)
    if ((selectedVC is not active) or (selectedVC_sizeFirst > totalQuantum))
        selectedVC_deficitCounter ← totalQuantum
        selectedVC ← Next active VC
        totalQuantum ← selectedVC_deficitCounter + selectedVC_quantum
    totalQuantum = totalQuantum − selectedVC_sizeFirst
    Transmit packet from selectedVC
    if ((There are no packets in the queue of selectedVC) or
        (The flow control does not allow transmitting from selectedVC))
        totalQuantum ← 0
```

Fig. 2. Pseudocode of the DRR-CA scheduler.

from each flow. This means that there will never be a need to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. As observed in [16], removing this hypothesis would entail operating at a complexity which can be as large as $O(N)$. Note that this restriction affects not only the weight assigned to the smallest flow, but to the rest of the flows in order to keep the proportions among them.

A. Hardware implementation of the DRR-CA scheduler

When a packet arrives at the head of a VC queue the scheduler receives a notification from the buffers. The DRR-CA scheduler just takes note of the packet size and actives the VC if there are enough flow control credits to transmit that packet. In order to select the next VC that can transmit packets, the scheduler must select the next active VC from the last selected VC in a list with all the VCs. The scheduler transmits packets from the same VC until that VC becomes inactive or there is no enough quantum to transmit more packets from that VC.

A possible way of implementing the mechanism that selects the next active VC would be to check sequentially all the VCs in the list starting from the contiguous position of the last selected VC (see Figure 3). However, in order to make this search in an efficient way, we have implemented it with a *barrel shifter* connected to an *order based bitonic network*. The barrel shifter rearranges the list in the correct order of search and the bitonic network finds the first active VC in a logarithmic number of cycles. The structure for this selector function is also shown in Figure 3.

IV. THE SCFQ-CA SCHEDULER

The Self-Clocked Weighted Fair Queuing (SCFQ) algorithm [13] is a variant of the Weighted Fair Queuing (WFQ) mechanism [11] which has a lower computational complexity. It defines fair queueing

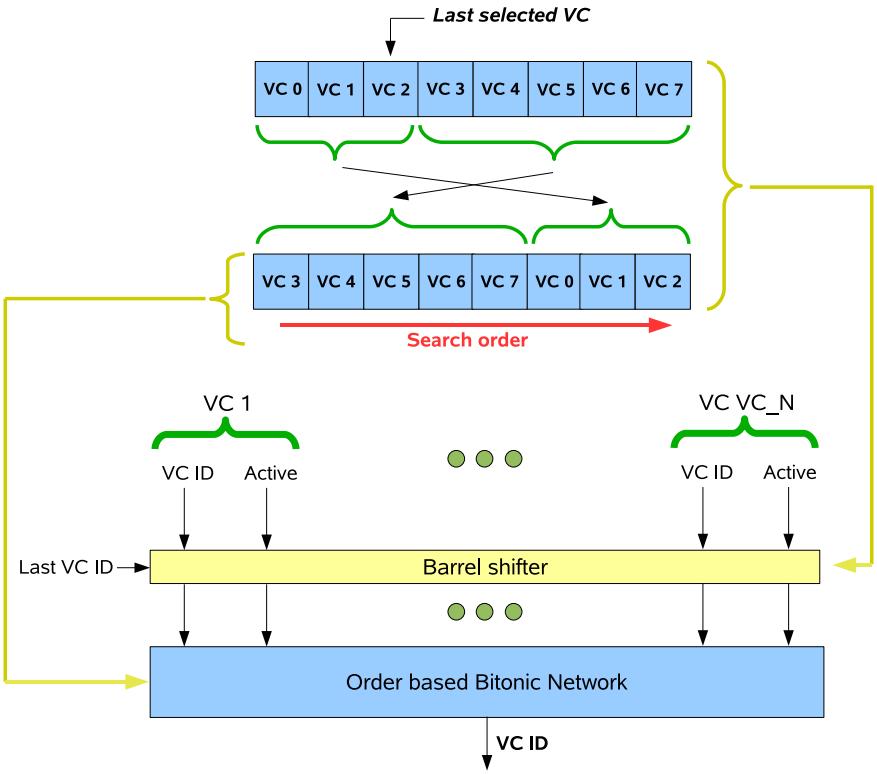

Fig. 3. Structure of the module that selects the next VC to transmit in the DRR-CA scheduler.

in a self-contained manner and avoids using a hypothetical queueing system as reference to determine the fair order of services like in the WFQ. This objective is accomplished by adopting a different notion of the virtual time. Instead of linking the virtual time to the work progress in the GPS system, the SCFQ algorithm uses a virtual time function which depends on the progress of the work in the actual packet-based queuing system. This approach offers the advantage of removing the computation complexity associated to the evaluation of the virtual time that may make WFQ unfeasible in high-speed interconnection technologies.

Therefore, when a packet arrives, SCFQ uses the service tag of the packet currently in service as the virtual time to calculate the new packet tag. Thus, in this case the service tag of the k^{th} packet of the i^{th} flow S_i^k, let L_i^k be its length and ϕ_i the weight assigned to its flow, is computed as

$$S_i^k = max\{S_i^{k-1}, S_{current}\} + \frac{L_i^k}{\phi_i}$$

As stated before, the SCFQ algorithm avoids the emulation of a GPS system to maintain the *virtual time*. This reduces the computational complexity of the tag calculation. Therefore, the computational complexity of the SCFQ algorithm is lower than the complexity of the WFQ algorithm. However, the computational simplification does not come without a cost: In some situations SCFQ behaves worse than WFQ and WF2Q. Figure 4 shows the pseudocode for the SCFQ algorithm.

The SCFQ-CA scheduler we have proposed adapts the original SCFQ algorithm to be used with a link-level flow control mechanism with a limited number of flows or VCs. Note that, $S_{current}$ is the service tag of the packet currently being transmitted and thus, the service tag of the packets that have already been transmitted is equal to or lower than $S_{current}$. Moreover, the service tag of the packets that have not already been transmitted are equal to or bigger than $S_{current}$. Therefore, if the k^{th} packet of the i^{th} VC arrives at an empty queue, the service tag is computed as:

$$S_i^k = S_{current} + \frac{L_i^k}{\phi_i}$$

On the other hand, if the k^{th} packet of the VC i arrives at a queue with more packets, the service

PACKET ARRIVAL (newPacket, flow):
$newPacket_{serviceTag} \leftarrow max(currentServiceTag, flow_{lastServiceTag}) + \frac{newPacket_{size}}{flow_{reservedBandwidth}}$
$flow_{lastServiceTag} \leftarrow newPacket_{serviceTag}$

ARBITRATION:
while (There is at least one packet to transmit)
    $selectedPacket \leftarrow$ Packet with the minimum $serviceTag$
    $currentServiceTag \leftarrow selectedPacket_{serviceTag}$
    Transmit $selectedPacket$
    if (There are no more packets to transmit)
        $\forall flow\ flow_{lastServiceTag} \leftarrow 0$
        $currentServiceTag \leftarrow 0$

Fig. 4. Pseudocode of the SCFQ scheduler.

tag is computed as:

$$S_i^k = S_i^{k-1} + \frac{L_i^k}{\phi_i}$$

This means that once that there is at least one packet in a VC queue, the value of the service tags of the packets that arrive after this first packet depends only on the value of the precedent service tags and not on the value of $S_{current}$ at the arrival time. Therefore, we can wait to stamp a packet p_i^k with its service time until the packet that is before it in the VC queue, p_i^{k-1}, is being transmitted. Note that at this time the $S_{current}$ is equal to S_i^{k-1}.

This allows us to simplify in a high degree the original SCFQ algorithm by storing not a service tag per packet, but a service tag per flow or VC. This service tag represents the service tag of the packet of the VC queue. Note that this makes much easier and simpler to modify this algorithm to take into account a link-level flow control mechanism. Each VC service tag is then computed as:

$$S_i = S_{current} + \frac{L_i^{first}}{\phi_i}$$

where L_i^{first} is the size of the packet at the head of the i^{th} VC.

The SCFQ-CA algorithm that we propose works in the same way as the SCFQ algorithm, except in the following aspects:

- We are going to consider VCs instead of flows in the SCFQ-CA algorithm because, as stated before, is the common trend in the last interconnection network proposals.
- Each active VC has associated a service tag.
- When a new packet arrives at a VC queue, a service tag is assigned only if the arrived packet is at the head of the VC and there are enough credits to transmit it.
- When a packet is transmitted, if there are enough credits to transmit the next packet, the VC service tag is recalculated.
- When a VC is inactive due to a lack of credits and receives enough credits to transmit again, a new service tag is assigned to the VC.

The resulting scheduling algorithm is represented in the pseudocode shown in Figure 5.

A. Symplifying the SCFQ-CA scheduler

The SCFQ-CA algorithm, as the original SCFQ algorithm and most shorted-priority algorithms, has the problem of the increasing tag values and the possible overflow of the registers used to store these values. Therefore, we propose a modification to the SCFQ-CA scheduler that makes impossible this overflow. This modification consists in subtracting the service tag of the packet currently being transmitted to the rest of service tags. If we consider only a tag per VC, this means to subtract the service tag of the VC to which the packet being transmitted belongs to the rest of VCs service tags.

This limits the maximum value of the service tags while still maintaining the absolute differences among their values. This also means that $S_{current}$ is

PACKET ARRIVAL(newPacket, VC):
if ($newPacket$ is at the head in the queue of VC) and
(The flow control allows transmitting from VC))
$VC_{serviceTag} \leftarrow currentServiceTag + \frac{VC_{sizeFirst}}{VC_{reservedBandwidth}}$

ARBITRATION:
while (There is at least one active VC)
$selectedVC \leftarrow$ Active VC with the minimum $serviceTag$
$currentServiceTag \leftarrow selectedVC_{serviceTag}$
Transmit a packet from $selectedVC$
if ((There are more packets in the queue of $selectedVC$) and
(The flow control allows transmitting from $selectedVC$))
$selectedVC_{serviceTag} \leftarrow currentServiceTag + \frac{selectedVC_{sizeFirst}}{selectedVC_{reservedBandwidth}}$
else
$selectedVC_{serviceTag} \leftarrow 0$
if (There are no active VCs)
$currentServiceTag \leftarrow 0$

Fig. 5. Pseudocode of the SCFQ-CA scheduler.

always equal to zero and thus,

$$S_i = \frac{L_i^{first}}{\phi_i}$$

Moreover, the service tags are limited to a maximum value max_S: $max_S = \frac{MTU}{min_\phi}$ where MTU is the maximum packet size and min_ϕ is the minimum possible weight that can be assigned to a VC. The resulting SCFQ-CA scheduling algorithm is represented in the pseudocode shown in Figure 6. Note that this last modification adds the complexity of subtracting to all the service tags a certain value each time a packet is scheduled. This makes this modification feasible in hardware only when a few number of VCs is considered, which is the common trend in the last interconnection network proposals.

B. Hardware implementation of the SCFQ-CA scheduler

When a new packet arrives at the SCFQ-CA scheduler, apart from taking note of the packet size and activating the VC if there are enough flow control credits to transmit that packet, this scheduler must calculate the packet service tag. As stated before, we have solved the problem of the possible overflow of the service tags. Moreover, this modification entails a simplification of the service tag computation.

In order to decide which is the next packet to be transmitted, the SCFQ-CA algorithm must choose the packet from the active VC with the smallest service tag. In order to do this in an efficient way, we have employed a bitonic network, which obtains the selected VC in log(#VCs) cycles. The structure of the selector module is shown in Figure 7.

V. The Deficit Table scheduler

We have proposed a new table-based scheduling algorithm that works properly with variable packet sizes [17]. We have called this algorithm Deficit Table scheduler, or just DTable scheduler, since it is a mix between the previously proposed table-based schedulers and the DRR algorithm. Our scheduler works in a similar way than the DRR algorithm but instead of serving packets of a flow in a single visit per frame, the quantum associated to each flow is distributed throughout the entire frame. Note that we have also considered the possibility of a link-level flow control mechanism when defining this scheduler.

This new table-based scheduler defines an arbitration table in which each table entry has associated a flow identifier and an *entry weight*, which is usually expressed in flow control credits in networks with a credit-based link-level flow control (like AS and IBA). Moreover, each flow has assigned a *deficit*

PACKET ARRIVAL(newPacket, VC):
if ($newPacket$ is at the head in the queue of VC) and
(The flow control does allow transmitting from VC))
$VC_{serviceTag} \leftarrow \frac{VC_{sizeFirst}}{VC_{reservedBandwidth}}$

ARBITRATION:
while (There is at least one active VC)
$selectedVC \leftarrow$ Active VC with the minimum $serviceTag$
$currentServiceTag \leftarrow selectedVC_{serviceTag}$
Transmit packet from $selectedVC$
$\forall$ active VC
$VC_{serviceTag} \leftarrow VC_{serviceTag} - currentServiceTag$
if ((There are more packets in the queue of $selectedVC$) and
(The flow control does allow transmitting from $selectedVC$))
$selectedVC_{serviceTag} \leftarrow \frac{selectedVC_{sizeFirst}}{selectedVC_{reservedBandwidth}}$

Fig. 6. Pseudocode of the improved SCFQ-CA scheduler.

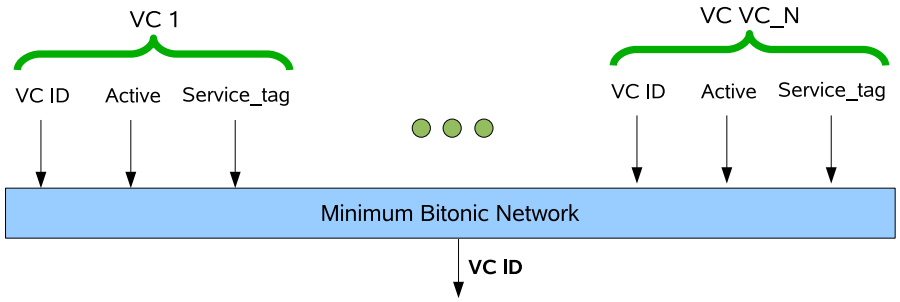

Fig. 7. Structure of the selector module for the SCFQ-CA scheduler.

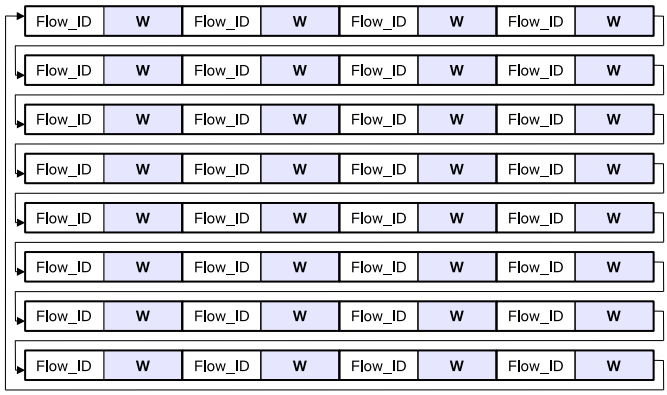

Fig. 8. Example of an arbitration table with 32 entries for the DTable scheduler.

counter that is set to 0 at the beginning. Figure 8 shows an example of an arbitration table with 32 entries.

When scheduling is needed, the table is cycled through sequentially until an entry assigned to an active flow is found. A flow is considered active when it stores at least one packet and the flow control allows that flow to transmit packets. When a table entry is selected, the *accumulated weight* is computed. The accumulated weight is equal to the sum of the deficit counter for the selected flow and the current entry weight. The scheduler transmits as many packets from the active flow as the accumulated weight allows. When a packet is transmitted, the accumulated weight is reduced by the packet size.

The next active table entry is selected if the flow becomes inactive or the accumulated weight becomes smaller than the size of the packet at the head of the queue. In the first case, the remaining accumulated weight is discarded and the deficit counter is set to zero. In the second case, the unused accumulated weight is saved in the deficit counter,

representing the weight that the scheduler owes the queue.

This behavior, already considering VCs instead of flows, is represented in the pseudocode shown in Figure 9. Note that when using the scheduling algorithm the bandwidth assigned to the i^{th} flow ϕ_i with an arbitration table of N entries is:

$$\phi_i = \frac{\sum_{j=0}^{J} weight_j}{\sum_{n=0}^{N} weight_n}$$

where J is the set of table entries assigned to the i^{th} flow and $weight$ is the entry weight assigned to a table entry.

In order to keep the computational complexity low, we set the minimum value that a table entry can have associated to the MTU of the network. This is the smallest value that ensures that there will never be necessary to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. This means that each time an entry from an active flow is selected, at least one packet is going to be transmitted from that flow. Note that this consideration is also made in the DRR algorithm definition [27]. Note also that in the IBA table-based scheduler this issue is solved by rounding up to a whole packet the remaining weight in a table entry.

A. Converting the AS and IBA table schedulers to a DTable scheduler

As stated before, AS and IBA employ table-based schedulers in the egress links to provide QoS. In this section, we show how to implement the DTable scheduler in these technologies modifying as little as possible their specifications.

1) InfiniBand: Each table entry of the IBA scheduling mechanism specifies a VC identifier and a weight. Therefore, the difference between the IBA table and the DTable arises when the size of the packet at the head of the selected VC is bigger than the remaining amount of information to transmit from that VC. In the InfiniBand case, the packet is transmitted exhausting the remaining weight, but in the DTable case, other table entry is selected, and the remaining weight is stored for future use in the VC deficit counter.

Therefore, in order to implement the DTable scheduler in IBA, it is only necessary to add the deficit mechanism. This means to add a deficit counter to each VC and the logic to store and load the remaining weight. Note that these counters are set to zero at the beginning and are modified dynamically by the scheduler itself during the scheduling process, and thus they do not require any user configuration.

2) Advanced Switching: As stated before, the AS arbitration table consists in a list of VC identifiers without any weight assigned to each entry as it is the case in the DTable scheduler. Therefore, apart from adding the hardware to manage a deficit counter per VC, we must indicate in some way the weight assigned to each table entry.

The simplest way of implementing the DTable scheduler would be to assign all the table entries the same fixed weight. However, this approach limits a lot the configuration posibilities of the DTable scheduler. Therefore, we have proposed three other possibilities to fully implement the DTable scheduler in AS but modifying as little as possible the AS specification [19]:

a) Using the 3-bit reserved field: This possibility consists in employing the 3-bit reserved field of each table entry to assign a weight to each entry. The problem of this implementation is that this field only allows us to specify a weight between 0 and 7, and thus, several considerations must be made. First of all, as stated before, the entry weight must represent at least the value of the MTU. Therefore, a weight of 0 is not going to be used, and thus, we propose to consider the weight 0 as 1, the weight 1 as 2, etc. This allows us to specify a weight between 1 and 8 with the 3-bit field.

Moreover, in AS, the MTU can be up to 34 flow control credits (2176 bytes). Obviously, it is not possible to represent directly a value of at least 34 with just 3 bits. Therefore, when using the 3-bit reserved field to assign a weight to each entry, each weight unit will represent a weight equivalent to a certain number of flow control credits m. Therefore, when an entry is selected its weight must be translated into its value in flow control credits:

$$tableEntry.weight \leftarrow (tableEntry.value + 1) \times m$$

b) Modifying the arbitration table format: Other possibility is to modify the structure of the arbitration table in order to dedicate a higher num-

```
while (There is at least one active VC)
    if ((selectedVC is not active) or (selectedVC_sizeFirst > accumulatedWeight))
        selectedVC_deficitCounter ← accumulatedWeight
        tableEntry ← Next table entry assigned to an active VC
        selectedVC ← tableEntry_VCIdentifier
        accumulatedWeight ← selectedVC_deficitCounter + tableEntry_weight
    accumulatedWeight = accumulatedWeight − selectedVC_sizeFirst
    Transmit packet from selectedVC
    if ((There are no packets in the queue of selectedVC) or
        (The flow control does not allow transmitting from selectedVC))
        accumulatedWeight ← 0
```

Fig. 9. Pseudocode of the DTable scheduler.

ber of bits to the entry weight. Specifically, we propose to use two bytes per table entry, and use 5 bits for the VC identifier and up to 11 for the entry weight. This number of bits is high enough to directly employ it for storing the entry weight:

$$tableEntry.weight \leftarrow tableEntry.value$$

c) Using only one weight per VC: The third possibility that we propose is to associate the same weight to all the entries assigned to a VC. Therefore, we only need to specify a table weight per VC instead of per table entry. This requires, however, an additional structure to configure a weight per VC. When a new table entry is selected, the accumulated weight is computed as:

$$tableEntry.weight \leftarrow weight_{selectedVC}$$

B. Hardware implementation of the DTable scheduler

When a new packet is notified to the DTable scheduler, it just takes note of the packet size and actives the VC if there are enough flow control packets to transmit that packet (it makes the same as the DRR-CA scheduler). As in the DRR-CA case, this scheduler transmits from the same selected VC until the VC becomes inactive or the remaining weight entry is not enough to transmit the packet at the head of the VC queue. In order to select a new VC to transmit from, the arbitration table must be looked over sequentially searching for the next active entry and skipping those entries that refer to a VC without packets or credits to transmit. Although the checking of each entry can be made with very simple computational units, in the worst case all the table must be looked over in order to find the next active entry.

In order to make the process faster, several entries of the table can be read simultaneously at the expense of increasing the silicon area and probably the cycle time. This algorithm also requires the memory necessary to store the arbitration table. However, this algorithm has not the problem of the increasing tag value and does not need mathematical division units to calculate any packet tag of sorted priority algorithms.

The arbitration table can be stored in specialized memory blocks, like the SRAM block that can be found in most FPGAs models, or in an array of registers. A possible way to read several entries simultaneously in an efficient way is to split the register array or memory block in several subblocks and read one entry of each of these subblocks in the same cycle. We have called the number of simultaneous table entries read in a single cycle the *parallelization grade*.

Figure 10 shows the structure of the mechanism that we have implemented to obtain the next active table entry. First of all we read a certain number of consecutive table entries from the last selected table entry equal to the parallelization grade. The next cycle, we check if any of those entries refers to an active VC. At the same time, the next ‘parallelization grade’ entries are read. When the mechanism realizes that at least one entry is active in the set of table entries, the process stops and a bitonic network is employed to calculate which is the first active entry in the subblock.

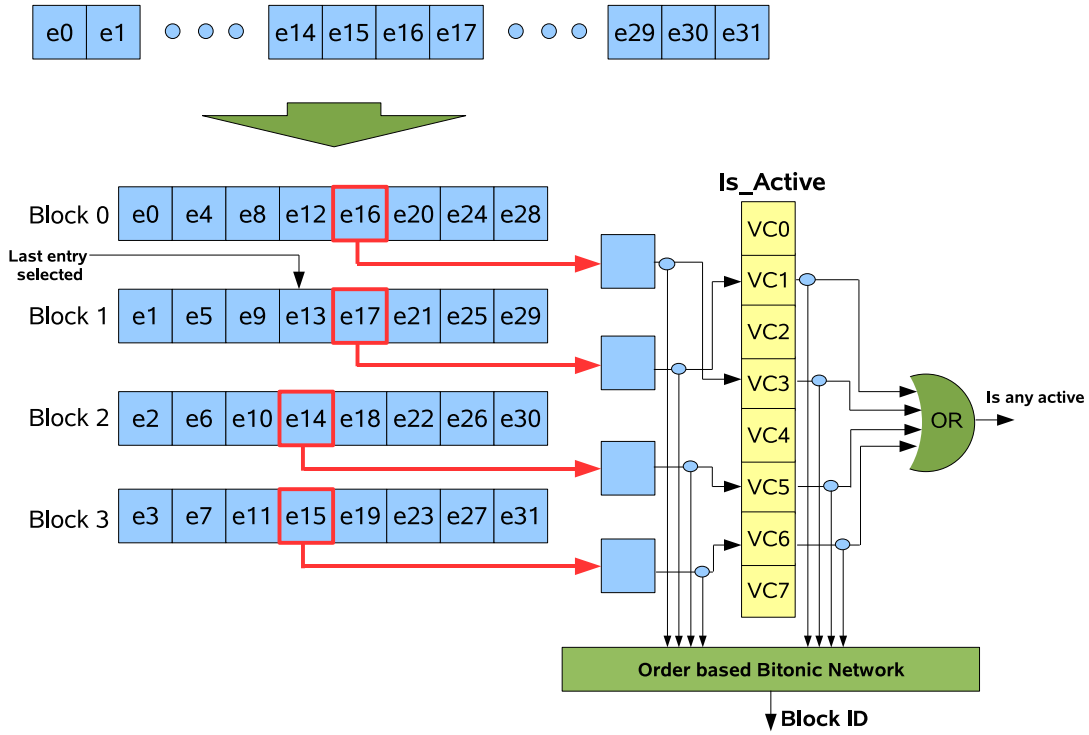

Fig. 10. Structure of the selector module for the parallel table scheduler.

TABLE I

ARBITRATION TIME IN CYCLES FOR SEQUENTIAL AND PARALLEL IMPLEMENTATIONS OF THE DTABLE SCHEDULER.

Scheduler	Number of cycles
Table (Sequential search)	$[1 \rightarrow \#Entries] + 2$
Table (Parallel search)	$[1 \rightarrow \frac{\#Entries}{Parallel\_Grade}] + log_2(Parallel\_Grade) + 3$

Table I shows the number of cycles required to make the arbitration decision in both cases, when the table is cycle through sequentially or, when various entries are processed at the same time. Note that in the DTable scheduler case, the number of cycles required to complete the arbitration is variable and depends on how far from the last selected entry is the next selected entry. When the load of the network is low, more cycles will be probably required in average to find the next table entry. When the load of the network is high, most VCs will be active anytime, and thus the average number of cycles will be very small.

VI. HARDWARE ESTIMATES

In this section we analyze the implementation and computational complexity of the DRR-CA, SCFQ-CA and DTable schedulers. We have modeled these schedulers using Handel-C language [7] and employed the DK design suite tool from Celoxica in order to obtain hardware estimates on silicon area and arbitration time. Note that the code that we have designed can actually be used to implement the DRR-CA and SCFQ-CA schedulers in a Field Programmable Gate Array (FPGA) or, if the appropriate conversion is made, in an Application Specific Integrated Circuit (ASIC). However, this has not been the objective of our work. Therefore, we have tried to implement the schedulers in an efficient way, but they could have been probably implemented more efficiently. Our objective has neither been to obtain explicit values for the silicon area nor for the arbitration time of each scheduler. In fact, these values are very dependent on the specific FPGA or the implementation technology employed. We are more interested in the relative differences on silicon

area and arbitration time for the different schedulers and the effect of some design parameters like the number of VCs or the MTU.

A. Handel-C and the DK design suite

As stated before, we have employed the Handel-C language to model and obtain hardware estimates for the different schedulers that we have considered. Handel-C is essentially an extended subset of the standard ANSI-C language, specifically extended for being used in hardware design (see Figure 11).

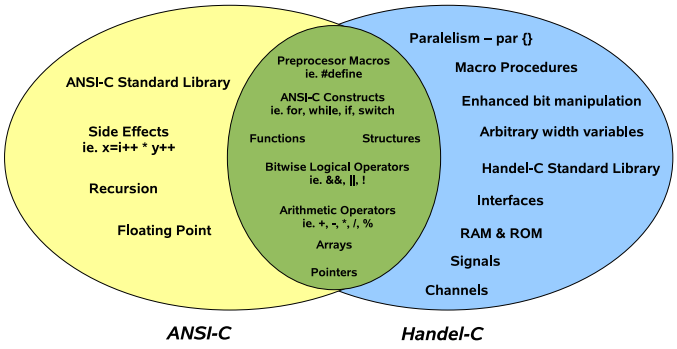

Fig. 11. ANSI-C / Handel-C comparison.

Handel-C's level of design abstraction is above Register Transfer Level (RTL) languages, like VHDL [2] and Verilog [21], but below behavioral. In Handel-C each assignment infers a register and takes one clock cycle to complete, so it is not a behavioral language in terms of timing. The source code completely describes the execution sequence and the most complex expression determines the clock period.

A comparison of Handel-C with RTL languages shows that the aims of these languages are quite different. RTL languages are designed for hardware engineers who want to create sophisticated circuits. They provide all constructs necessary to craft complex, tailor made hardware designs. By choosing the right elements and language constructs in the right order, the specialist can specify every single gate or flip-flop built and manipulate the propagation delays of signals throughout the system. On the other hand, RTL languages expect that the developer knows about low-level hardware and requires him continuously thinking about the gate-level effects of every single code sequence.

In contrast to that, Handel-C is not designed to be a hardware description language, but a high-level programming language with hardware output. It does not provide highly specialized hardware features and allows only the design of digital synchronous circuits. Instead of trying to cover all potentially possible design particularities, its focus is on fast prototyping and optimizing at the algorithmic level. The low-level problems are hidden completely, all the gate-level decisions and optimization are done by the compiler so that the programmer can focus his mind on the task he wants to implement. As a consequence, hardware design using Handel-C resembles more to programming than to hardware engineering.

Handel-C closely corresponds with a typical software flow and provides the essential extensions required to describe hardware. These extensions include flexible data widths, parallel processing and communications between parallel threads. Sequential by default, Handel-C has a *par* construct. When a block of code is qualified by *par*, statements are executed concurrently and synchronized at the block end. This simple construct allows for the expression of mixed sequential and parallel flows in compact and readable code.

The Handel-C compiler comes packaged with the Celoxica DK design suite. The DK design suite supports several output targets:

- Debugger: The debugger provides in-depth features normally found only in software development. These features include breakpoints, single stepping, variable watches, and the ability to follow parallel threads of execution. The hardware designer can step through the design just like a software design system using this approach.
- EDIF: The second output target is the synthesis of a netlist for input to place and route tools. Place and route is the process of translating a netlist into a hardware layout. This output allows the design to be translated into configuration data for particular chips. When compiling the design for a hardware target, Handel-C emits the design in Electronic Design Interchange Format (EDIF).
- RTL (VHDL and Verilog): The RTL output preserves the hierarchy of the Handel-C source code allowing experienced engineers to verify at the RTL level. The compiler generates RTL with appropriate syntax and attributes for leading third party synthesis tools, timing simulators and ASIC design flows.

In order to obtain the hardware estimates in which we are interested:

1) We have modeled in Handel-C a full egress queuing system, including the scheduler.
2) We have validated the schedulers employing the simulation and debugging functionality of the DK design suite.
3) We have isolated the scheduler module in order to obtain estimates without influence of other modules.
4) We have obtained the EDIF output for a Virtex 4 FPGA from Xilinx [35].

A cycle count is available from the Handel-C source code: Each statement in the Handel-C source code is executed in a single cycle in the resulting hardware design and thus, the number of cycles required to perform a given function can be deduced directly from the source code. Moreover, an estimate of gate count and cycle time is generated by the EDIF Handel-C compiler. The cycle time estimate is totally dependent on the specific target FPGA, in this case the Virtex 4 [35], which is one of the last FPGA models provided by Xilinx [34]. However, as our objective is to obtain relative values instead of absolute ones, we consider that this approach is good enough to be able to compare the complexity in terms of silicon area and scheduling time of the different schedulers. Figure 12 reflects the design flow that we have followed.

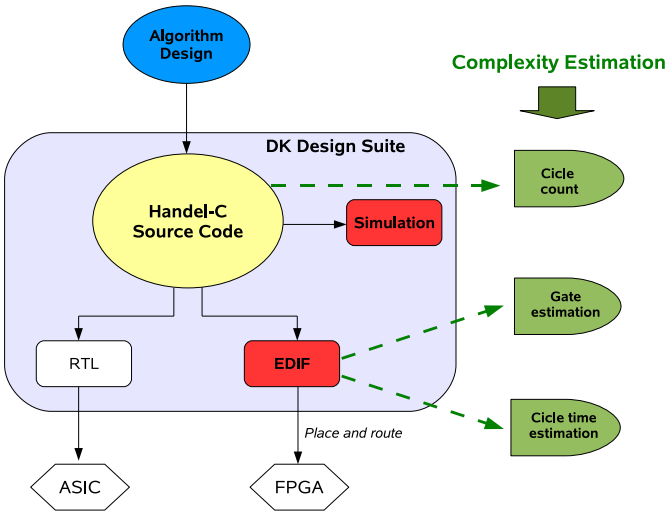

Fig. 12. Design flow with DK employing Handel-C.

B. Modelling the egress queuing system

As stated before, in order to model the different schedulers, we have previously modeled a full egress link queuing system that could be part of an endnode or switch. We have done this in order to be able to test the rightness of our implementation. Figure 13 shows the different modules that compound the egress queuing system and their interactions. These modules are:

- **Traffic generator:** We need a traffic load in order to test the schedulers. We have developed a Constant Bit Rate (CBR) traffic generator in order to feed the VCs. We can assign each VC with a different traffic generator configured to produce packets at a different rate and with different packet size.
- **Buffers:** The buffers module is the responsible of managing the packets stored in each VC queue. It tracks the available space in each queue, notifies the scheduler the arrival of new packets, and frees space in the queues when packets are transmitted.
- **Transmitter:** The transmitter module injects into the egress link the packets that the scheduler indicates and deletes the information of those packets in the buffers.
- **Scheduler:** The scheduler module is the most important part to our objective. Its main function is to decide the next packet to be transmitted from an active VC. In order to do so, it keeps track of the set of active VCs by monitoring the packet at the head of each queue and the available number of flow control credits. Moreover, it consumes the flow control credits required by each transmitted packet. When a scheduling decision has finished it notifies that fact to the transmitter.
- **Flow controller:** The flow controller tracks the number of available flow control credits of each VC.
- **Credit generator:** Only one egress queuing system has been modeled, and thus in order to keep the system transmitting packets we need to renew the consumed flow control credits with a flow control credit generator module.

An advantage of using Handel-C to model the egress queuing system and the schedulers is that it allows parameterizing the design in an easy way. Through the use of constants and compiler commands we can generate outputs (for simulation, EDIF, or RTL targets) with, for example, variable number of VCs and packet MTU considered. In

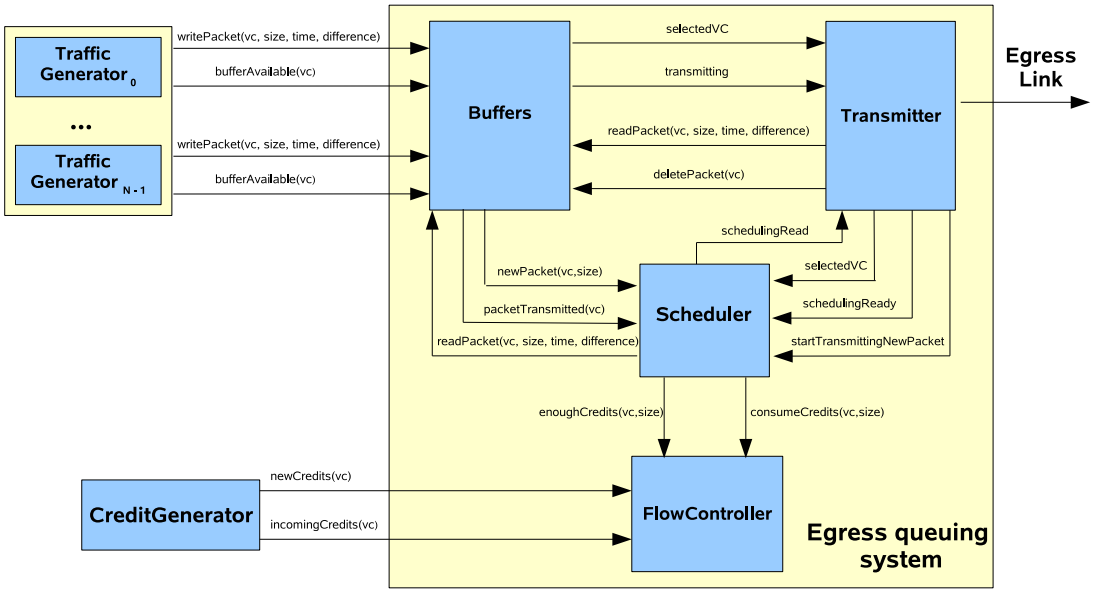

Fig. 13. Egress link queuing system modules.

order to simplify the design, we have considered power of two values for the number of VCs and MTU. Moreover, we have considered packets to be of an integer number of flow control credits.

C. Hardware estimates for the DRR-CA and SCFQ-CA schedulers

Note that, in order to calculate the packet service tag a division operation is required, which is not a simple arithmetical unit. Handel-C offers a divisor operand that calculates the result in one cycle (as all the Handel-C statements). This operand makes the division very short in terms of number of cycles but, it makes the cycle time very long, and thus it makes the arbitration time quite long. Therefore, we have also implemented a version of the SCFQ-CA scheduler that employs a mathematical division unit that performs the division in several cycles. Specifically, it takes a number of cycles equal to the length of the operators plus one. This second version reduces the cycle time and thus, the arbitration time. However, the division requires much more cycles to be performed. It even requires more time to be performed because the cycle time is not reduced in the same proportion as the number of cycles is increased. We have called the SCFQ-CA version that performs the division in one cycle '*atomic SCFQ-CA*'. On the other hand, we have called the SCFQ-CA version that performs the division in several cycles '*segmented SCFQ-CA*'.

The advantage of the atomic SCFQ-CA is that it calculates the time tag in only one cycle, and thus it takes for processing a new packet, the same time than the DRR-CA scheduler, and as we will see, also than the table scheduler. This makes very easy to compare both schedulers, because it is only necessary to confront the silicon area and arbitration time. However, in the segmented SCFQ-CA case processing a new packet takes much more time, and without a full model of a switch the effect over the overall performance of this longer time is not easy to be measured. We include this option in this study because it is a possibility that must be taken into account, but the comparison with the rest of schedulers is not so clear like in the atomic SCFQ-CA case.

As stated before, once that the schedulers have been validated through simulation with the debugger functionality of the DK design suite, we have isolated the scheduler module in order to compile it for the EDIF output. In this way the hardware estimates obtained, like the cycle time, are not going to be influenced by the rest of modules. Table II shows the number of cycles required by the DRR-CA and SCFQ-CA schedulers to perform the arbitration. Therefore, the arbitration time depends on the cycle time and on the number of VCs (VC_N in the table).

Figure 14 shows how the increment in the number of VCs and the MTU affects the silicon area and the arbitration time of the DRR-CA and SCFQ-CA

TABLE II
ARBITRATION TIME IN CYCLES FOR THE DRR-CA AND SCFQ-CA SCHEDULERS.

Scheduler	Number of cycles
DRR-CA	$log_2(VC\_N) + 3$
Atomic SCFQ-CA	$log_2(VC\_N) + 2$
Segmented SCFQ-CA	$log_2(VC\_N) + 2$

schedulers (atomic and segmented). Specifically, it shows the increment in these complexity indices respect the simplest case for each scheduler (2 VCs and a MTU of 2). When varying the number of VCs, we have used a MTU of 32 and when varying the MTU we have considered 8 VCs.

Regarding the effect of the number of VCs, Figure 14 shows that this number influences dramatically the silicon area and arbitration time required by the DRR-CA and SCFQ-CA schedulers. Note that in the case of the arbitration time, the increment is due to both, the increase in the cycle time and the increase in the number of cycles required to compute the arbitration. Note that the X axis is in logarithmic scale, thus a linear growth in data plot actually means a logarithmic data growth, and an exponential growth in data plot actually means a linear data growth.

On the other hand, regarding the effect of the MTU, Figure 14 shows that the increase in silicon area and time when increasing the MTU is not so important if compared with the effect of the number of VCs. The atomic variant of the SCFQ-CA scheduler is the most affected by this parameter. Increasing the MTU from 2 to 64 increases the silicon area required by this scheduler 70% and the arbitration time 37%. The reason of this is that the value of the MTU affects the size of the division operation required to calculate the SCFQ-CA service tag and thus, it affects in a higher degree the atomic version of the SCFQ, which requires a lot of silicon area and increases in a high degree the cycle time in order to perform the division in a single cycle.

Figure 15 shows the same results than Figure 14 except that in this case, the increment is relative to the silicon area and arbitration time required by the DRR-CA scheduler with 2 VCs (when varying the number of VCs) and a MTU of 2 (when varying the MTU). This allows us to compare the silicon area and the arbitration time required by the different schedulers for different design parameters.

Figure 15 shows, as expected, that the DRR-CA scheduler is the simplest scheduler in terms of silicon area and arbitration time. On the other hand, the atomic version of the SCFQ-CA scheduler requires much more silicon area and arbitration time than the DRR-CA or the segmented SCFQ-CA schedulers. Figure 15 also shows that the segmented SCFQ-CA scheduler requires also much more silicon area than the DRR-CA scheduler. However, the difference in arbitration time is not so big. Finally, this figure shows that the difference among the atomic SCFQ-CA scheduler and the other two scheduler increases with the MTU.

D. Hardware estimates for the DTable scheduler

In order to obtain hardware estimates of the DTable scheduler we have considered, apart from the number of VCs and the MTU, the number of table entries and the parallelization grade as design parameters. Moreover, we have also calculated hardware estimates to compare the original AS table with the possible implementations of the DTable scheduler shown in Section V-A.2.

Figure 16 shows the difference in silicon area and arbitration time of the different table possibilities. Note that the increment in time refers to both, the minimum and maximum arbitration time required by the scheduler. Specifically, the figure shows the increment in silicon area and time respect the original AS table scheduler. In all the cases, a table of 128 entries with a parallelization grade of 16, 8 VCs, and a MTU of 32 is considered. Figure 16 shows that employing a fixed weight for all the table entries (*FixedW*), which solves the problem of the original table scheduler with variable packet size, only requires 10% more silicon area than the original AS table scheduler (*Original*).

However, in order to have a greater flexibility when configuring the DTable scheduler, we can

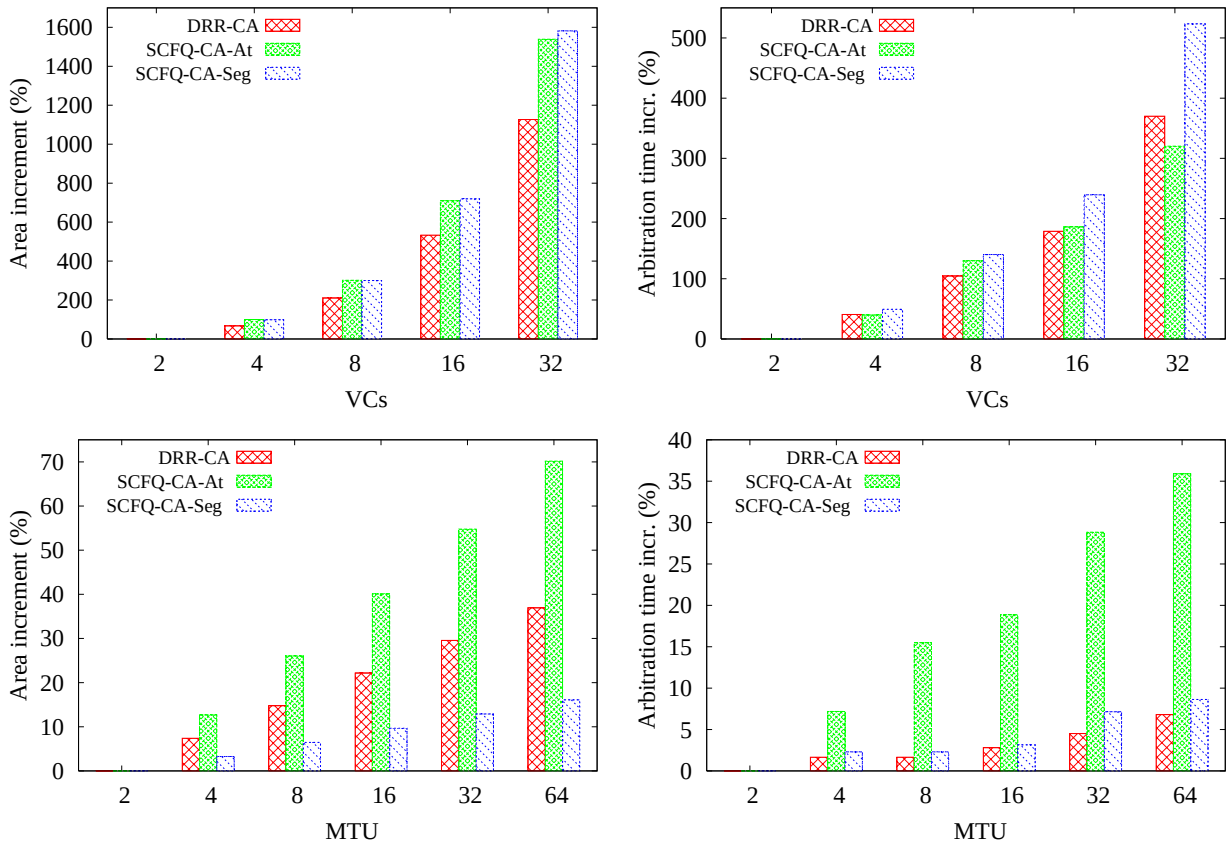

Fig. 14. Effect of the number of VCs and MTU over the silicon area and arbitration time required by the DRR-CA and the SCFQ-CA schedulers.

choose between using a weight per each VC (*W-VC*), using the three reserved bits of each table entry (*3-bits*), or using two bytes to store the VC identifier and the table entry (2-bytes).

Figure 16 shows that the 2-bytes option is the most demanding one. This option requires 80% more silicon area than the original AS table compared with the 35% of the 3-bit option. Moreover, the arbitration time is slightly higher (0.85%) than in the rest of the cases, which have the same arbitration time, and thus the increment is 0%. In the rest of this work we will show statistics of the 2-bytes DTable option because is the worst case for all the table implementations. Moreover, this is the possibility that provides the best flexibility and granularity.

Figure 17 shows the effect of the number of VCs over the complexity of a DTable with 128 entries, a parallelization grade of 16, and a MTU of 32. Specifically, it shows the increment in silicon area and arbitration time required respect the 2-VC case. This figure shows that this parameter affects in a high degree the silicon area required and, when the number of VCs is very high, a little the arbitration time. However, the effect is not so dramatic as in the DRR-CA and SCFQ-CA cases. Note that from 2 to 8 VCs the arbitration time is the same, and thus the increment is 0%. The reason because the number of VCs does not affect as much the complexity as in the DRR or SCFQ cases is that in the DTable case the scheduling is made over the arbitration table and not over a list of VCs, like in the DRR-CA case where we search for the next active VC, or the SCFQ-CA, where we search for the VC with the minimum service tag.

Figure 18 shows the effect of the MTU value over the complexity of a DTable with 128 entries, a parallelization grade of 16, and 8 VCs. Specifically, it shows the increment in silicon area and arbitration time required respect the 2-MTU case. This figure shows that the MTU is almost irrelevant for the silicon area and arbitration time required by this scheduler.

Figure 19 shows the effect of the number of table entries over the complexity of a DTable with

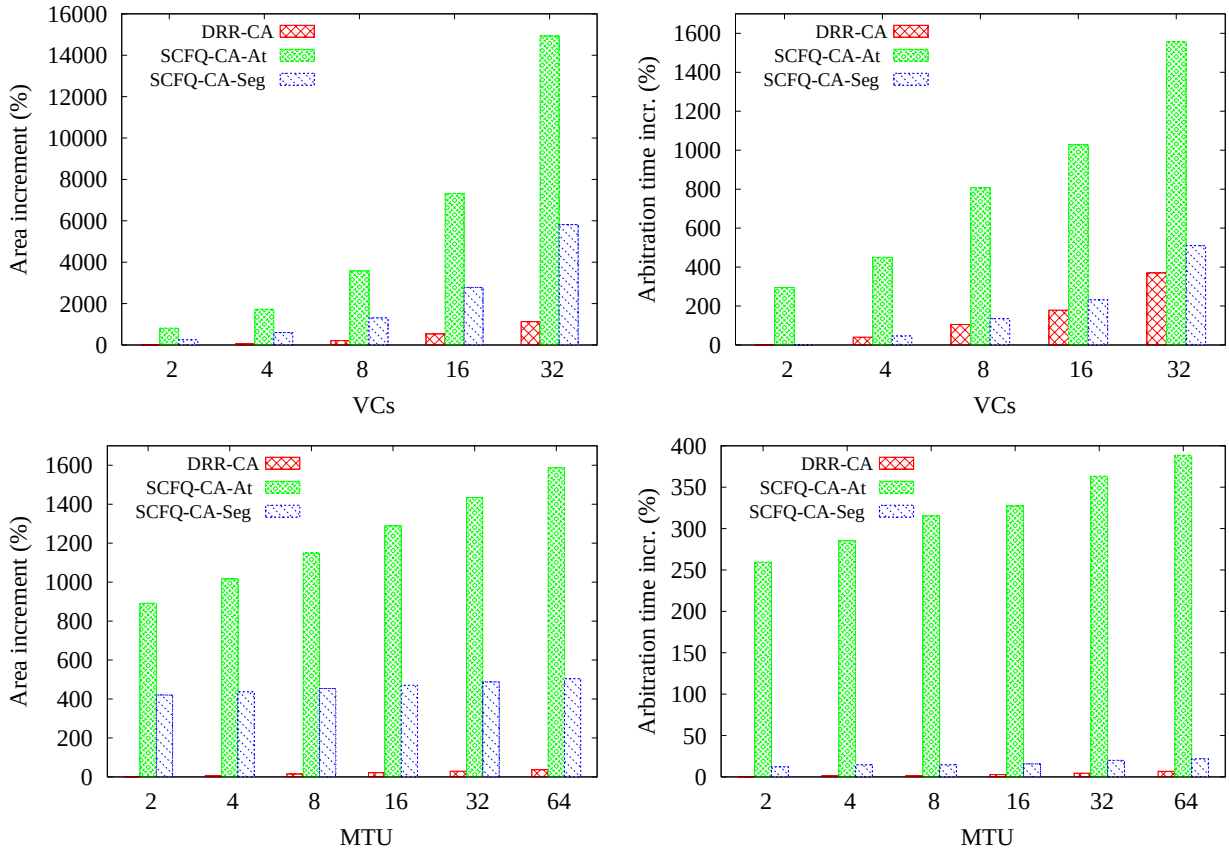

Fig. 15. Comparison of the silicon area and arbitration time required by the DRR-CA and the SCFQ-CA schedulers.

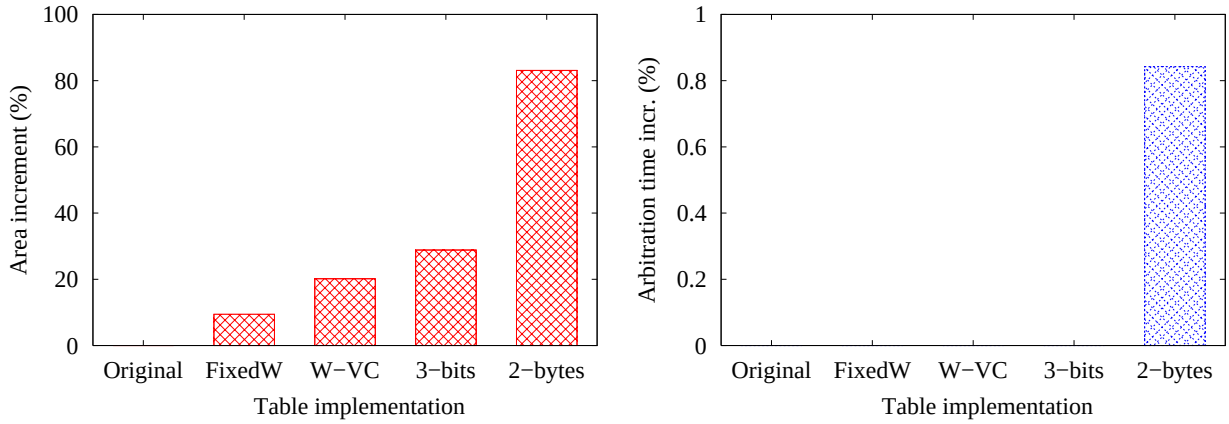

Fig. 16. Complexity comparison of the different possible implementations of the DTable scheduler (8 VCs, 128 entries, a paralelization grade of 16, and a MTU of 32).

a parallelization grade of 16, when the MTU is 32 and there are 8 VCs. Specifically, this figure shows the increment in silicon area, cycle time, and minimum and maximum time required to perform the arbitration respect the silicon area and minimum time required in the 32-entry case. This parameter affects in a high degree both the silicon area and the arbitration time. The increment in the silicon area is due to the increment in the space required to store the arbitration table and the extra logic to handle it. The increment in the arbitration time is due to the increment in the cycle time, but also to the extra number of cycles required to process a bigger table. Specifically, the increment in the cycle time determines the increment in the minimum time required to make the arbitration. Note that we

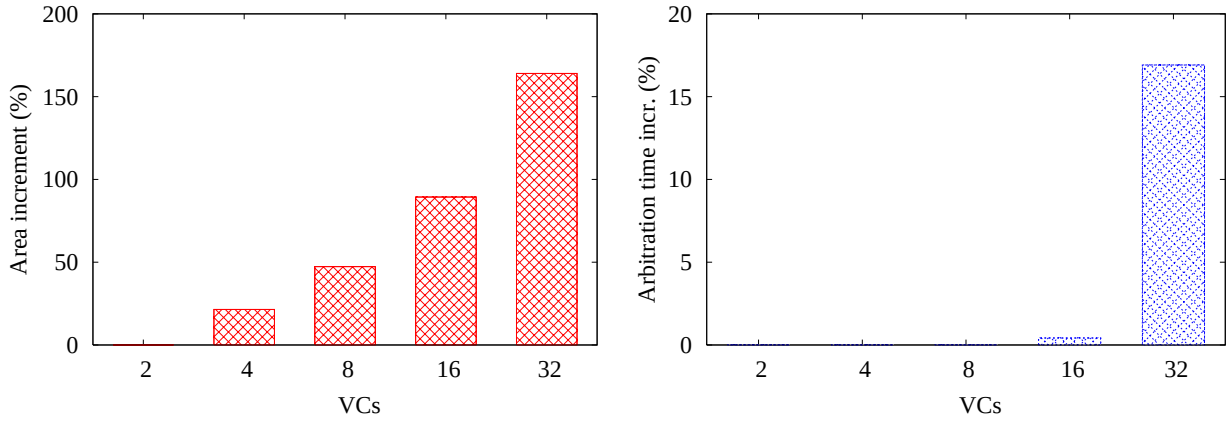

Fig. 17. Effect of the number of VCs over the silicon area and arbitration time required by the DTable scheduler (128 entries, a paralelization grade of 16, and a MTU of 32).

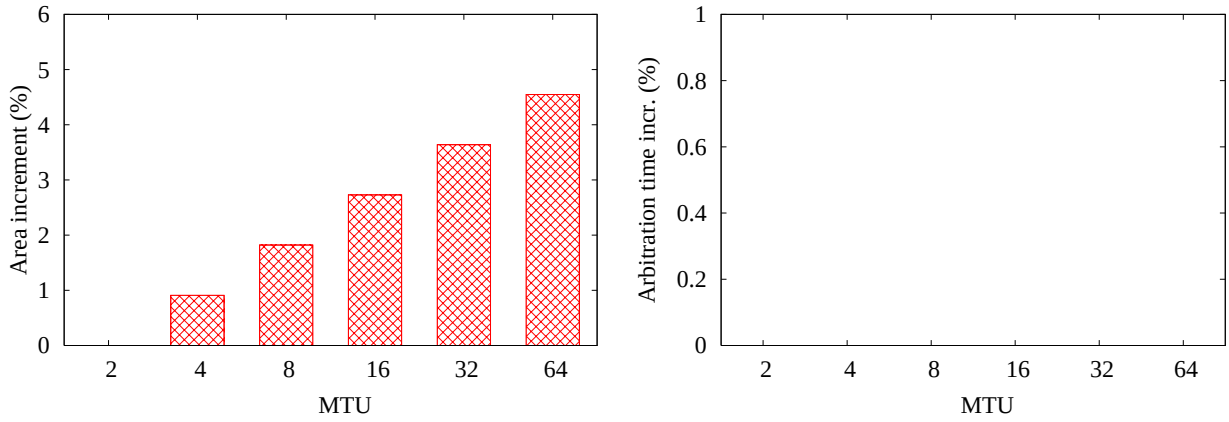

Fig. 18. Effect of the number of the MTU over the silicon area and arbitration time required by the DTable scheduler (8 VCs, 128 entries, and a paralelization grade of 16).

use the same parallelization grade in all the cases and thus, the same minimum number of cycles is required to perform the arbitration (see Table I). On the other hand, the maximum number of required cycles increases with the table size and thus, the maximum required time increases dramatically.

A way to reduce the arbitration time is to increase the parallelization grade. Figure 20 shows the effect of this parameter over a DTable of 128 entries, 8 VCs, and a MTU of 32. Specifically, this figure shows the increment in silicon area, cycle time, and minimum and maximum time required to perform the arbitration, respect the silicon area and minimum time required when the parallelization grade is 1 (sequential search). This figure shows that increasing the parallelization grade also increases in a high degree the silicon area required. This extra area is not so exacerbate when we increase only a bit the parallelization grade. However, if we increase the value of this parameter a lot, the silicon area increases much faster. Given a certain number of entries (128 in this case), the effect of increasing the parallelization grade is to reduce the maximum number of cycles required to perform the arbitration at the cost of increasing the minimum number of cycles required (see Table I). This effect is shown in Figure 20. However, this figure shows that increasing too much the parallelization grade affects in a negative way both the minimum and maximum arbitration time because of the increment in the cycle time.

Until now we have shown the individual effects of varying the value of the different design parameters over a basic configuration of a 2-bytes DTable with 128 entries, a parallelization grade of 16, a MTU of 32, and 8 VCs. Figure 21 shows a more general picture in which we observe the effect of varying the number of VCs for every table size. At the same

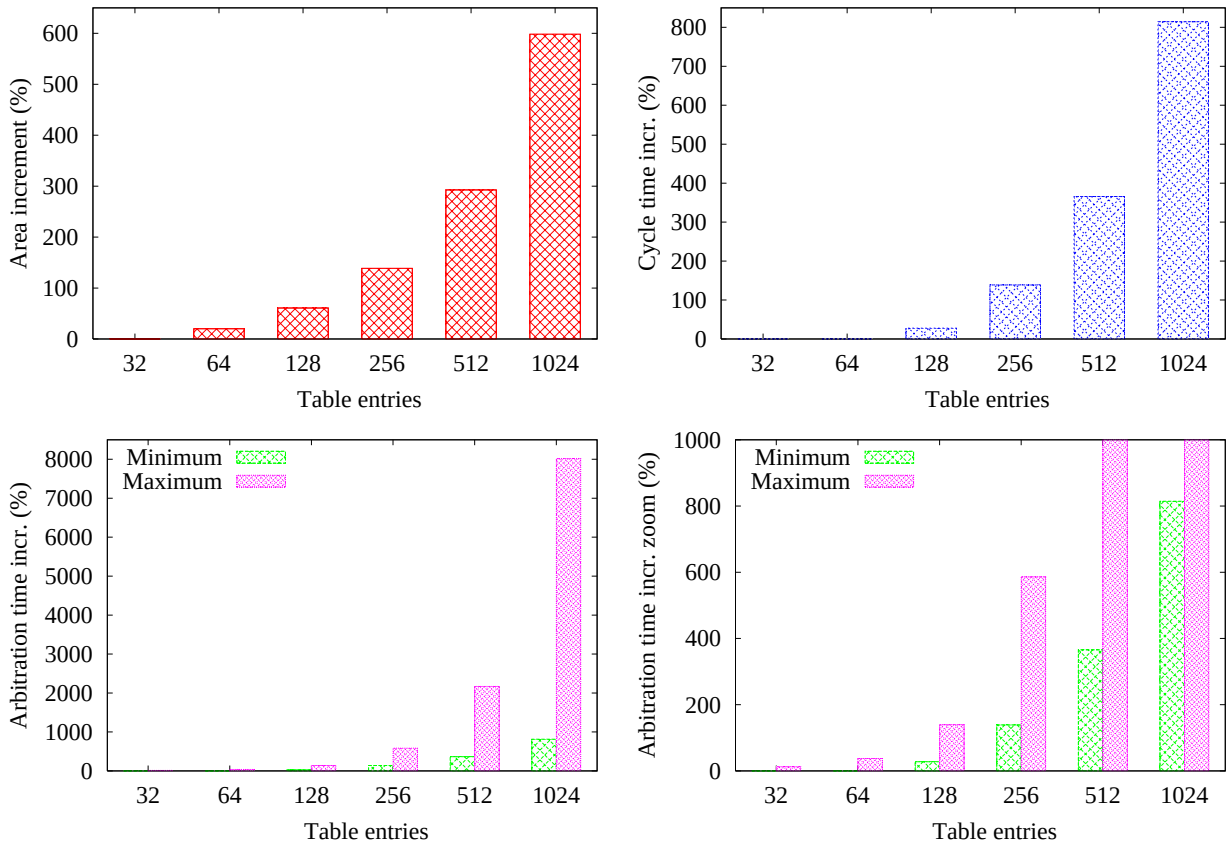

Fig. 19. Effect of the number of table entries over the silicon area and arbitration time required by the DTable scheduler (8 VCs, a paralelization grade of 16, and a MTU of 32).

time we vary the parallelization grade in order to keep constant and equal to 16 the number of cycles required to process all the table entries (number of entries / parallelization grade = 16). Note that even with this last consideration, the number of cycles is not the same in each combination of number of entries and parallelization grade (see Table III). The increments shown are respect to DTable with 32 entries and 2 VCs.

Figure 21 shows that when the number of table entries grows, the silicon area required increases dramatically due to the accumulated effect of the increment on the table size and the parallelization grade. However, even increasing the parallelization grade the arbitration time also grows a lot due to the increment on the cycle time. A smaller arbitration time could be achieved increasing more the parallelization grade, however, this would increase even more the silicon area required. Figure 21 also shows that the number of VCs is only relevant for the arbitration time for small arbitration table sizes. When the arbitration table has lots of entries, the number of VCs does not affect the cycle time and thus, the arbitration time.

E. Comparing the DRR-CA and the SCFQ-CA schedulers with the DTable scheduler

In the previous sections we have shown how the different design parameters affect the complexity, in terms of silicon area and arbitration time, of the DRR-CA, SCFQ-CA, and DTable schedulers. In this section we are going to compare the complexity of these schedulers.

Figures 22 and 23 show a comparison of the silicon area and arbitration time required with different number of VCs for the different schedulers and, in the case of the DTable scheduler, different number of table entries (we have also kept number of entries / parallelization grade = 16). Note that not all the possible combinations of number of VCs and number of table entries make sense. If we have a lot of VCs, we will probably need more table entries to accommodate appropriately all those VCs. Note, for example, that in an extreme case where we have 32

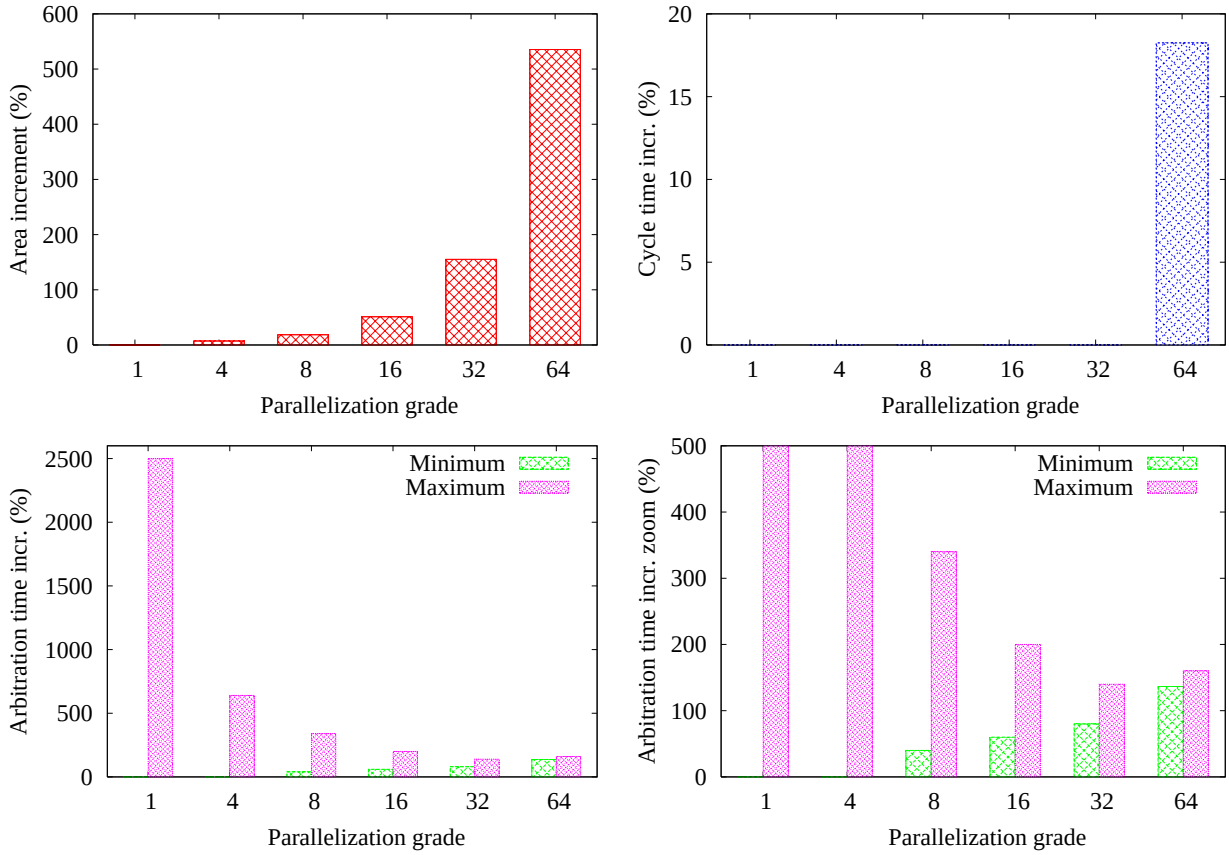

Fig. 20. Effect of the parallelization grade over the silicon area and arbitration time required by the DTable scheduler (8 VCs, 128 entries, and a MTU of 32).

TABLE III

COMBINATION OF VALUES FOR THE TABLE ENTRIES AND PARALLELIZATION GRADE AND ARBITRATION TIME IN CYCLES.

Number of table entries	Parallelization grade	Arbitration time (cycles)
32	4	6 - 13
64	8	7 - 14
128	16	8 - 15
256	32	9 - 16
512	64	10 - 17
1024	128	11 - 18

VCs and 32 entries, we should assign each VC to a given table entry and we would not be able to make any latency differentiation. On the other hand, if we have very few VCs, it would be a waste of resources to employ a lot of table entries. Therefore, we have only shown the combination of 2 and 4 VCs with 32, 64, and 128 table entries, and 16 and 32 VCs with 256, 512, and 1024 table entries. For the 8-VC case we show the interaction with the possible table sizes. Moreover, we have split the data in two separate figures in order to show them more clearly. Both figures show the increment on silicon area and minimum and maximum arbitration time required respect to the DRR-CA with 2 VCs.

Figure 22 shows the comparison of the schedulers for a small number of VCs (2-8) and a small number of table entries (32-128). This figure shows that, as expected, the DRR-CA is the simplest scheduler in terms of both, silicon area and arbitration time. The atomic version of the SCFQ-CA scheduler is the most demanding implementation also in both aspects. Regarding the DTable scheduler and the segmented version of the SCFQ-CA scheduler, Figure 22 shows that in general the DTable scheduler

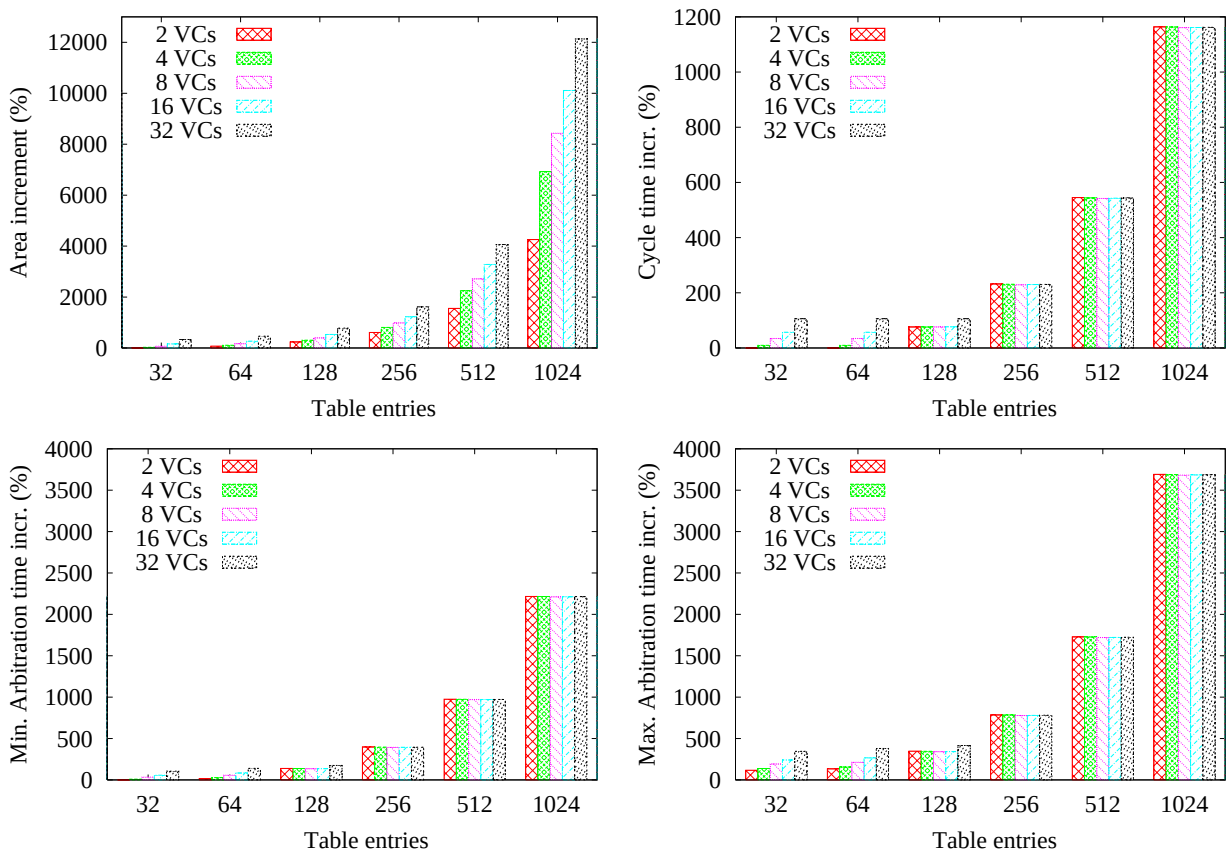

Fig. 21. Silicon area and arbitration time increment for the combined effect of the number of table entries and number of VCs for the DTable scheduler (paralelization grade of 16, and a MTU of 32).

requires less silicon area than the segmented SCFQ-CA scheduler. On the other hand, the SCFQ-CA scheduler is faster than the DTable scheduler. However, as stated before, in this comparison we do not take into account the extra time required by the segmented SCFQ-CA scheduler to compute the service tag.

Figure 23 shows the comparison of the schedulers for a high number of VCs (8-32) and a high number of table entries (256-1024). This figure shows that the DTable scheduler is the most complex when the number of table entries is 1024. When the table has 512 entries, only if it has 32 VCs it requires less silicon area than the atomic SCFQ-CA scheduler. When the DTable arbitration table has 256 entries this scheduler requires less silicon area than the atomic SCFQ-CA case. The time required in this case by the atomic SCFQ-CA case is in general higher than the minimum time required by the DTable scheduler but smaller than the maximum time. In almost all the cases the segmented SCFQ-CA case and the DRR-CA schedulers require less silicon area than the DTable with a size between 256 and 1024 entries.

VII. Conclusions and Future Work

In this work we have proposed specific implementations for three fair queueing scheduling algorithms: the DRR-CA, SCFQ-CA, and DTable schedulers. We have optimized their implementation in order to fullfill the complexity constrains in high-performance networks. We have proposed implementation improvements over their basic definition for the SCFQ and DTable schedulers. Moreover, we have performed a complexity comparison study of these three scheduling algorithms. In order to do so we have implemented the schedulers in Handel-C and obtained hardware statistics employing the DK design suite tool.

We have studied the complexity in terms of silicon area and time required to perform the scheduling. We have obtained hardware estimates for these indices taking into account different values for some design parameters. We have considered the number

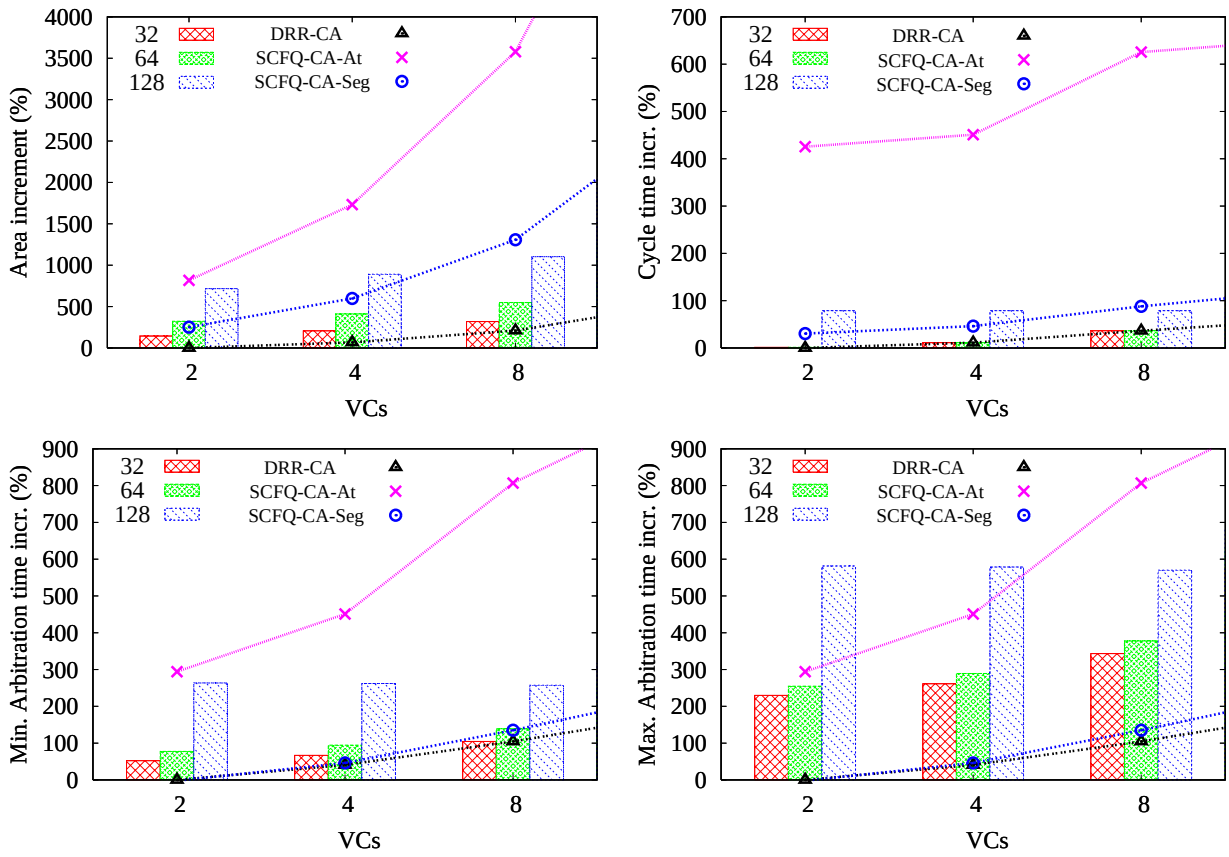

Fig. 22. Silicon area and arbitration time comparison of the different schedulers with a small number of VCs (DTable with a paralelization grade of 16, and a MTU of 32).

of VCs and the MTU in all the cases. Moreover, for the DTable scheduler we have also considered the size of the table in terms of table entries and the parallelization grade, which is the number of table entries that we read each cycle. Furthermore, we have also compared the complexity of the different implementation options for the DTable scheduler.

The hardware estimates that we have obtained have shown that the cost of modifying the original AS table to handle in a proper way variable packet sizes is very small (around 10% increment in silicon area). If we want to fully implement the DTable scheduler we only need to double the silicon area required. This increment compared with the entailed to increase the number of table entries or the parallelization grade is quite small.

The hardware estimates obtained also show that, as expected, the DRR-CA scheduler is the simplest one. The DTable scheduler is in general the most complex option when implementing large arbitration tables, which are required when there are a high number of VCs. However, the DTable scheduler can be a good option, at least in terms of silicon area, when a small number of table entries is implemented (32-256) if compared with the SCFQ-CA scheduler.

REFERENCES

[1] Advanced Switching Interconnect Special Interest Group. *Advanced Switching core architecture specification. Revision 1.0*, December 2003.

[2] P. J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann; 2nd edition, 2002.

[3] J. Bennett and H. Zhang. WF2Q: Worst-case fair weighted fair queueing. *INFOCOM*, 1996.

[4] J. C. R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Trans. Netw.*, 5(5):675–689, 1997.

[5] Y. Bernet. A Framework for Differentiated Services. Internet draft 2275, Internet Engineering Task Force, May 1998.

[6] S. Blake, D. Back, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. Internet Request for Comment RFC 2475, Internet Engineering Task Force, December 1998.

[7] Celoxica. *Handel-C Language Reference Manual for DK4*, 2005.

[8] H. Chao and X. Guo. *Quality of Service Control in High-Speed Networks*. Wiley, 2001.

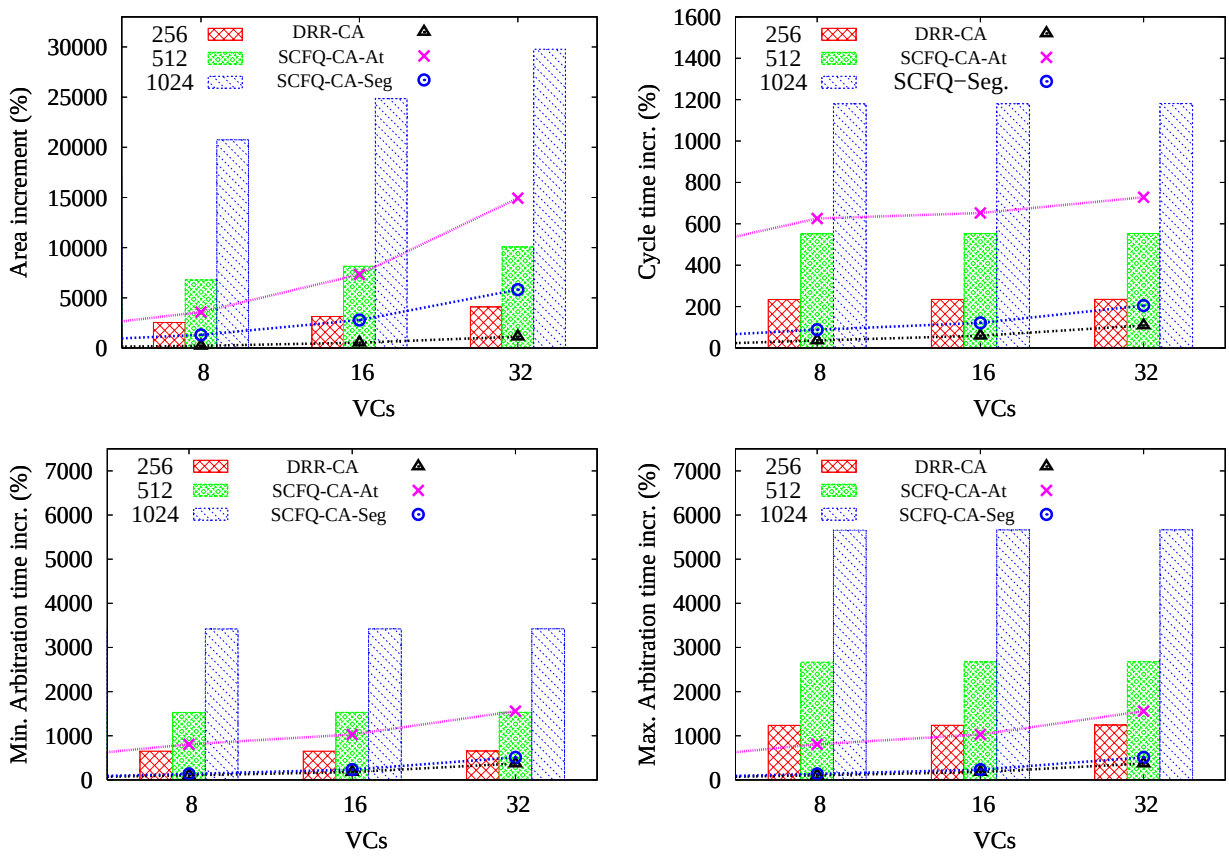

Fig. 23. Silicon area and arbitration time comparison of the different schedulers with a high number of VCs (DTable with a paralelization grade of 16, and a MTU of 32).

[9] H. M. Chaskar and U. Madhow. Fair scheduling with tunable latency: A round-robin approach. *IEEE/ACM Transactions on Networking*, 11(4):592–601, 2003.

[10] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter. Interconnect-aware coherence protocols for chip multiprocessors. In *ISCA*, pages 339–351. IEEE Computer Society, 2006.

[11] A. Demers, S. Keshav, and S. Shenker. Analysis and simulations of a fair queuing algorithm. In *SIGCOMM*, 1989.

[12] A. Charny et al. Supplemental information for the new definition of EF PHB (Expedited Forwarding Per-Hop-Behavior). RFC 3247, March 2002.

[13] S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOM*, 1994.

[14] A. G. Greenberg and N. Madras. How fair is fair queuing. *J. ACM*, 39(3):568–598, 1992.

[15] InfiniBand Trade Association. *InfiniBand architecture specification volume 1. Release 1.0*, October 2000.

[16] S. S. Kanhere, H. Sethu, and A. B. Parekh. Fair and efficient packet scheduling using elastic round robin. *IEEE Transactions on Parallel and Distributed Systems*, 2002.

[17] R. Martínez, F. J. Alfaro, and J.L. Sánchez. Decoupling the bandwidth and latency bounding for table-based schedulers. *International Conference on Parallel Processing (ICPP)*, August 2006.

[18] R. Martínez, F. J. Alfaro, and J.L. Sánchez. Implementing the Advanced Switching minimum bandwidth egress link scheduler. *IEEE International Symposium on Network Computing and Applications (IEEE NCA06)*, July 2006.

[19] R. Martínez, F. J. Alfaro, and J.L. Sánchez. Studying several proposals for the adaptation of the DTable scheduler to advanced switching. *International Symposium on Parallel and Distributed Processing and Applications (ISPA)*, December 2006.

[20] P. L. Montessoro and D. Pierattoni. Advanced research issues for tomorrow's multimedia networks. In *International Symposium on Information Technology (ITCC)*, 2001.

[21] S. Palnitkar. *Verilog HDL*. Prentice Hall PTR; 2 edition, 2003.

[22] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1993.

[23] K. I Park. *QoS in Packet Networks*. Springer, 2005.

[24] Sven-A. Reinemo, T. Skeie, T. Sødring, O. Lysne, and O. Tørudbakken. An overview of QoS capabilities in InfiniBand, Advanced Switching interconnect, and Ethernet. *IEEE Communications Magazine*, 44(7):32–38, 2006.

[25] J. Rexford, A. G. Greenberg, and F. Bonomi. Hardware-efficient fair queueing architectures for high-speed networks. In *INFOCOM (2)*, pages 638–646, 1996.

[26] R. Seifert. *Gigabit Ethernet: Technology and Applications for High-Speed LANs*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[27] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM*, pages 231–242, 1995.

[28] V. Sivaraman. *End-to-Ent delay service in high speed packet networks using Erliest Deadline First Scheduling*. PhD thesis, University of California, 2000.

[29] D. Stiliadis. *Traffic scheduling in packet-switched networks: Analysis, design, and implementation*. PhD thesis, University of California, 1996.

[30] D. Stiliadis and A. Varma. Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. *SIGMETRICS Perform. Eval. Rev.*, 24(1):104–115, 1996.

[31] D. Stiliadis and A. Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 1998.

[32] J. S. Turner. New directions in communications (or which way to the information age). *IEEE Communications*, 24(10):8–15, October 1986.

[33] P. Vellore and R. Venkatesan. Performance analysis of scheduling disciplines in hardware. In *Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2004.

[34] Xilinx, Inc. `http://www.xilinx.com`.

[35] Xilinx. Virtex-4 family overview. Fact sheet DS112 (v2.0), June 2007.

[36] Jun Xu and Richard J. Lipton. On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms. *IEEE/ACM Trans. Netw.*, 13(1):15–28, 2005.

[37] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks, 1995.

[38] Q. Zhao and J. Xu. On the computational complexity of maintaining gps clock in packet scheduling. In *IEEE INFOCOM*, March 2004.